DOCTOR OF PHILOSOPHY

Case based design of knitwear

Paul Richards

2013

Aston University

Case Based Design Of Knitwear

Paul Richards

Doctor of Philosophy

Aston University

October 2012

Aston University

Case Based Design of Knitwear

Paul Richards

Doctor of Philosophy

2012

In the developed world we are surrounded by man-made objects, but most people give little thought to the complex processes needed for their design. The design of hand knitting is complex because much of the domain knowledge is tacit. The objective of this thesis is to devise a methodology to help designers to work within design constraints, whilst facilitating creativity.

A hybrid solution including computer aided design (CAD) and case based reasoning (CBR) is proposed. The CAD system creates designs using domain-specific rules and these designs are employed for initial seeding of the case base and the management of constraints.

CBR reuses the designer's previous experience. The key aspects in the CBR system are measuring the similarity of cases and adapting past solutions to the current problem. Similarity is measured by asking the user to rank the importance of features; the ranks are then used to calculate weights for an algorithm which compares the specifications of designs.

A novel adaptation operator called rule difference replay (RDR) is created. When the specification to a new design is presented, the CAD program uses it to construct a design constituting an approximate solution. The most similar design from the case-base is then retrieved and RDR replays the changes previously made to the retrieved design on the new solution.

A measure of solution similarity that can validate subjective success scores is created. Specification similarity can be used as a guide whether to invoke CBR, in a hybrid CAD-CBR system. If the newly resulted design is sufficiently similar to a previous design, then CBR is invoked; otherwise CAD is used.

The application of RDR to knitwear design has demonstrated the flexibility to overcome deficiencies in rules that try to automate creativity, and has the potential to be applied to other domains such as interior design.

Key words: case based reasoning, computer-aided design, heuristics.

To Adele and Leo:

I watch you grow

you'll learn much more

than I'll ever know.

Acknowledgements

I would like to thank my supervisor Dr Anikó Ekárt for giving me the opportunity to undertake this fascinating research, and more importantly for her invaluable guidance, advice and support during the project.

I am grateful to Sirdar Spinning Ltd for providing not only some of the funding for this work, but also access to data, and experts in the field. This work was supported by a CASE studentship through EPSRC.

The love and support of my family has been particularly important. My wife Sarah and my children Adele and Leo have been very patient during the course of this work. I am grateful for my parents for their important support, both emotional and financial.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Imagination is more important than knowledge." - Albert Einstein

In the developed world we are surrounded by man-made objects, but most people give little thought to the often complex and labour intensive processes needed for the design of these objects. A particular design process (that of hand knitting) is explored here in order to build an understanding of how design thinking can be supported by automated systems. This has the potential to ease the burden of work on a designer, improve the quality of designs, and also reduce costs.

1.1 Design problems

This project is an exploration of how to support a designer to solve a complex design problem. Design is an activity with both creative and technical aspects. In order for products to be commercially successful, it is often important that they are aesthetically pleasing or stylish. However, designed objects are also functional, and they must be capable of fulfilling the purpose required by the end user. Usually, designs have to adhere to constraints; for example they may have limits on their size, weight or cost. Also, many problem domains have rules that the design must adhere to, in order for it to be valid.

Designers are typically given a so-called brief or specification, which defines the limits of their work. It is the presence of rules and a specification that differentiates design from art. A software system can support a designer by helping them to work within the limits of the specification, but also allowing them the necessary creative freedom.

In order to support a design process, a software system must allow the designer to be creative. Designers must have the freedom to explore different solutions in their mind's eye, and to view the design from different levels of detail. Also, the system must support the technical aspects of the design by reducing the burden of checking that the designer normally performs, so that the designer has the reassurance that their work is technically feasible.

Design is a competitive activity and designers take pride in the individual nature of their work. However, design is to some extent derivative as designers reuse aspects of existing designs. In fact, the creativity is derivative also: although this seems contradictory, innovation is copied and applied to new situations. So, a software system could assist the designer by being a repository of design knowledge, that the designer can reuse.

1.2 Knitwear design

Hand knitting has been employed to produce clothes since the thirteenth century or earlier [6], and even in modern times it is a popular leisure activity. Knitters work from patterns which consist of a mix of textual instructions and codes. Although knitting patterns look complex to the uninitiated, knitwear design comprises of an even more skilled set of activities. There is an initial creative phase where the designer studies fashion trends and formulates high-level ideas about the appearance of the garments. The designer will usually do a freehand (pencil and paper) sketch of the garment, capturing the shapes in their mind's eye.

The sketch will then be annotated with several important measurements, for example the overall length and width. The measurements are chosen so that the garment that is produced is wearable, and also contains the required shapes. There are conventions that are normally followed in knitwear design, for example most garments are symmetrical. This PhD concerns the design of the most frequent garments, i.e. sweaters and cardigans. These are normally knitted as separate pieces of fabric, and then sewn together. A very important convention is that the pieces must fit together, otherwise the garment will not be feasible. Thus, knitwear design has both creative and technical aspects.

The shapes used in knitwear drawings or sketches are a mixture of straight lines and curves. In cardigans and sweaters, curves are used for features such as a neckline, armhole and top of the sleeve. The curves can normally be described by simple mathematical models, as hand knitting is not a domain where things are expressed in high precision. The complexity in knitwear design arises because of the many combinations of options of shape that are possible. The design must contain the shapes that are dictated by fashion trends, but also be technically feasible as a knitted garment.

An automated system could assist knitwear design in several ways. The technical rules of knitting could be enforced by the system, to ensure the output is correct. It could provide visual feedback to changes so that the designer's need to be creative is supported. Ideally, it could also anticipate a designer's requirements. Also, a system should make it easy to re-apply the work done in previous designs to meet new design requirements. This has the potential to reduce the labour content of the design process, presumably also reducing costs.

Typically, computer-aided design (CAD) packages tend to be effective at supporting the mechanistic stages of design, and often provide rapid feedback, allowing the designer to visualise their work as it progresses. However, they are often poor at anticipating the needs of a designer, since automation is often "hard-coded" into the CAD program. So, a more flexible solution is sought for the knitwear problem.

1.3 Case based reasoning (CBR)

CBR is an artificial intelligence technique which aims to solve new problems by directly reusing the solutions to previously encountered problems. Many techniques work by reusing previous knowledge; however the difference is that CBR reuses specific snippets of knowledge, known as cases. It is distinct from other strategies that solve problems by using general rules or formulae.

CBR was inspired by research in cognitive science that suggested humans often solve everyday problems by remembering previous solutions to similar problems [7]. For example, a doctor might diagnose a patient by remembering a previous patient with similar symptoms, or a computer help-desk operator might resolve a problem by remembering a previous issue [8].

CBR systems contain a database or store of previously solved problems. There are many variants on CBR, but in general the technique works as follows. When a new problem is input into the system, it is compared against the problems in the store. The problem from the store that is the most similar to the new problem is then retrieved. The retrieved problem is then reused for the solution to the new problem. The new problem is retained with its new solution in the store for future use. The CBR system builds up experience as the case base grows, since the more problems are contained in the store, the more likely there will be one that is very similar to each new problem that is presented.

There are two main challenges in CBR: how to compare problems, and how to reuse solutions. Some method of measuring the similarity between problems is necessary in order to determine which of the problems in the store is the most similar to the current problem.

The second challenge is how to reuse the solution (that had previously been applied to the problem from the store) to solve the new problem. This is usually the most complex part of CBR, since although the two problems are similar, they are usually not identical and so the solution cannot often be reused without changes. Some CBR systems avoid this challenge by simply presenting the previous solution to the user and leaving it up to them how it is used; this is sometimes called CBR-lite [4, 9].

CBR is often cited as being useful for the sort of problems which are in areas not underpinned by strong theories. If there is a theory, formula or law which can be used to solve the problem, then CBR may not be necessary. However, many "real world" problems, for example solving computer faults in a help-desk, do not have such formulae and often rely on the intuition or experience of humans. CBR can be effective at storing and reusing this experience. Since hand knitting is such a "real world" problem, CBR has the potential to be used as a repository of the designer's knowledge.

1.4 Motivation and research objectives

This thesis seeks to explore solutions to a complex design problem: how to automate the knitwear design process, in a way which minimises the labour of the designer. The problem naturally lends itself to two types of solutions. The most obvious is a computer-aided design (CAD) program that uses straightforward rules to help the user construct a design. The rules would be necessary to ensure a sensible starting point: it would be unrealistically labour intensive to expect a user to create a design from nothing.

The second solution that this research will explore in detail is the use of case-based reasoning. Many design problems have a repetitive aspect, since designers reuse previous designs as sources of inspiration. Also, fashion trends repeat themselves from one garment to the next. So, CBR has the potential to act as a store of design expertise, and to re-apply this to new designs.

Both of these problem-solving approaches have obvious deficiencies. The chief disadvantage of the CAD approach is that it is difficult to construct rules to aid design. It is impossible to anticipate all of a designer's requirements, since there are large variety of designs that are possible. The difficulty with CBR is that it learns from experience, and so it requires a store of previous experience to work. This research is starting from a zero base; although some designs are accessible, they are not in an electronic format that would be amenable to CBR.

The strategy used is a combination of the two: a hybrid CAD-CBR system. This thesis is about exploring the issues involved with constructing a hybrid system, and experimenting with its efficacy. Such an exploration can give rise to many questions, such as:

i What is the most effective way of using a combination of CBR and CAD to automate the knitwear design process?

ii How can problems be compared?

iii How can the solutions to problems be reused?

iv How can we measure success? What constitutes a 'good' design?

v How much "experience" does a CBR system need to acquire before it becomes effective in the knitwear problem?

Knitting is an interesting domain for computer science research for a number of reasons. Firstly, little academic study has been made of hand knitting. Most of the literature concentrates on machine knitting, which has different characteristics. Secondly, it is a problem domain which has technical aspects: knitted garments have to obey rules in order to be valid, although there is some leeway in those rules. Thirdly, knitwear is a creative domain where success is subjective: there are no mathematical formulae which can tell you if a cardigan is beautiful! Finally, some aspects of knitwear design are repetitious. The same shapes and methods are used again and again; this makes it amenable to automation.

Finally, hand knitting is interesting because it is a realistic problem. All problems must have limits and therefore are underpinned by assumptions. However, many academic studies concentrate on 'toy problems', which are limited by the use of unrealistically strong assumptions. Therefore, it is sometimes difficult to see how the work on these toy problems can be extended to other areas. This issue does not apply to knitwear design, which is a messy "real-world" problem. One of the objectives of this thesis is to limit the assumptions behind the work so that the resulting system is as realistic as possible.

1.5 Thesis layout

The thesis is organised as follows.

Chapter 2: Case Based Reasoning. This chapter explains the concepts of CBR in detail, and provides a critical literature review of work in the area. It begins with an examination of CBR's early roots in cognitive science, then follows this with a discussion of the types of tasks that CBR is used for. Then, each of the challenges to constructing a CBR system is discussed in detail: case representation, retrieval, adaptation and case base maintenance. There is a discussion about hybrid systems, for example the combination of CBR and rule-based systems. The chapter ends with a discussion of the assumptions behind CBR, and its advantages and disadvantages.

Chapter 3: Design. This chapter reviews work in design studies. There then follows a short presentation of relevant topics in computer-aided design research. Knitwear design is then discussed, with a focus on hand knitting, although literature in this area is scarce. Finally, two categories of design process are defined (creative and mechanistic). The differences between the two categories are discussed, with reference to the examples of other work discussed in the chapter.

Chapter 4: The Knitwear Domain. This chapter provides the necessary background material about the problem domain of hand knitting. It begins with a high-level description of the issues in knitwear design, then discusses the types of shapes that are commonly used. As with all design domains, knitwear has constraints, and the nature of these is explained. Then, the more detailed concepts in knitwear design (stitches and stitch patterns) are explained. Finally, knitwear design is compared to other processes such as weaving or machine knitting.

Chapter 5: Knitwear CAD System. This chapter presents a detailed description of the computer-aided design system which allows designs to be created, viewed and edited. Initially, the requirements for the system are discussed, with reference to the existing design process (which is largely manual with little automation). The system has two main parts: questionnaire and sketch. The way in which the user completes the questionnaire and this generates a sketch is described, and the tools for editing the sketch are presented. There are two major complications in the knitwear sketches: managing constraints, and resizing. The way in which both of these issues is handled is explained in detail.

Chapter 6: Representing and comparing designs Both the CBR and CAD functionality require the storage and manipulation of data. This chapter explains how the data is represented; it begins with a discussion of what constitutes a good representation in this problem domain. Then, the implementation of the knowledge representation is briefly discussed. The knowledge representation must support the comparison of problems, in order that the most similar problem may be obtained. Similarity assessment is one of the most important challenges in CBR, and is addressed in SEACOP by comparing questionnaires using a weighted-sum algorithm, which is explained and justified. Finally, sketch similarity is concerned with measuring the similarity between solutions to the design problems presented here; this is intended to be used as an objective measure of the success of CBR and CAD.

Chapter 7: Adaptation Reuse of knowledge can take many forms, and its knitwear design aspects are discussed. This chapter presents a novel strategy for reusing design knowledge called rule difference replay (RDR). RDR works in processes which fulfil certain criteria, and these criteria are listed and the reasons for them justified. The application of RDR to knitwear design is then explained. An important part of this strategy is the method of mapping the features which are in common to two different designs. The discussion finishes with an explanation of how checks are performed to ensure the reuse was appropriate in order to produce a technically feasible design.

Chapter 8: Experiments This chapter presents the experiments that were performed on the hybrid CAD-CBR system. The hypotheses that the experiments are seeking to prove (or disprove) are stated. Then, the experimental methodology is outlined. Two experiments were performed: one uses the 'leave one out' methodology which is common in machine learning. Subsidiary to this is a Monte Carlo simulation which attempts to make predictions about how the CBR can learn from experience. The significance of the results of the experiments are analysed. There is an explanation of whether the evidence affirms or contradicts each of the hypotheses. Recommendations are made as to how the algorithms presented could form part of a coherent system.

The thesis then concludes with a synopsis of the main findings, and a brief discussion of how the work could be extended.

Chapter 2

Case Based Reasoning

> "Each problem that I solved became a rule which served afterwards to solve other problems." - Rene Descartes

2.1 What is CBR?

This section introduces the problem-solving methodology known as *case based reasoning* (CBR). After explaining the basic concepts of CBR, its theoretical and historical foundations are discussed. Then, the capabilities of CBR are explained via a discussion of the types of problems that CBR systems have been successfully applied to.

2.1.1 Introduction

Humans are typically taught to solve problems by other humans; initially their parents, and then their elders, teachers, peers or colleagues. Some problems are straightforward and a complete description of their solution can be given. However, many problems are more complex, involving several choices which are interdependent on each other; or even seemingly being impervious to logic at all, and supposedly solved using intuition. Complex tasks in this category include medical diagnosis, engineering design, and judicial decision-making.

People who are experienced in solving such complex problems often teach novices by giving them a series of *rules*. However, there is evidence (e.g. [7]) that when the learner becomes competent, many of the rules are gradually abandoned, and problems are solved by remembering previous occasions when solutions have been successful. For example, a doctor on encountering an ill patient may recall a previous instance of a patient with the same (or similar) symptoms. He or she may repeat the treatment given, possibly with some modifications, assuming that *similar problems require similar solutions*.

CBR is a problem-solving methodology which is inspired by human intellectual reasoning. CBR systems learn through experience by reusing the solutions to previously encountered problems, similar to

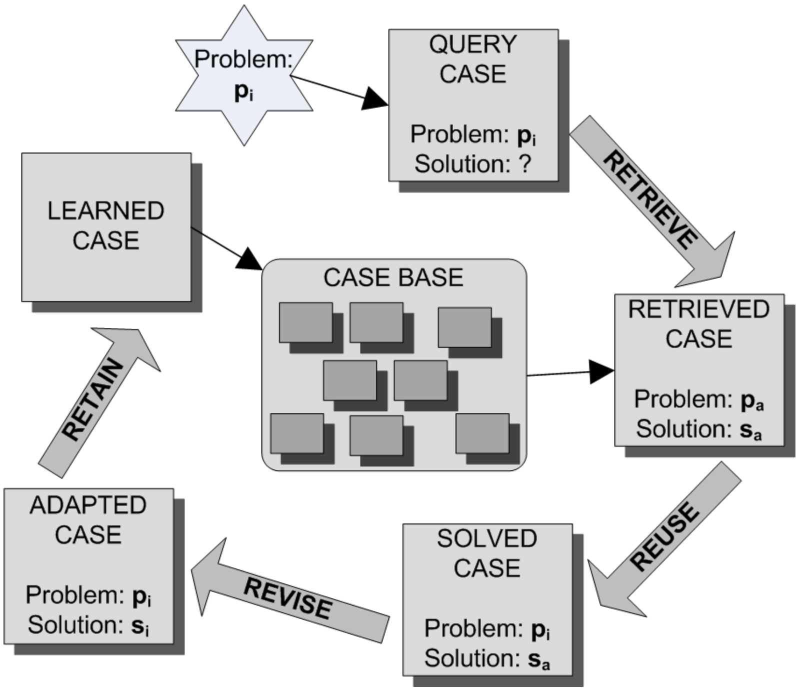


Figure 2.1: The 4RE cycle, adapted from [1]

the doctor encountering symptoms which are reminiscent of a previous patient. It is substantially different to other techniques because it uses examples to map problems to their solutions, rather than relying on rules, formulae or logic.

Although research in CBR began in the 1980s, it was influenced by earlier works in cognitive science and philosophy. Schank [7] proposed theories about how the human memory is organised, which argued that people learn from examples through failure. Early CBR systems (e.g. Cyrus [10]) were a proof of concept of problem-solving capabilities.

CBR research flourished during the 1990s, with extensive development of both theory and applications. CBR builds up a repository of previous successful solutions, called a *case base*. New problems are solved by finding and adapting a similar problem from the case base. Aamodt and Plaza's seminal paper [1] summarised the CBR process as a 4 step cycle, as shown in Figure 2.1. The cycle always starts with a new problem, p_i, of which a solution is required; this is known as the *query case* [11]. CBR proceeds by searching for a similar problem to p_i from the case base, labelled in figure 2.1 as p_a; the CBR system will *retrieve* p_a from the case base. Problem p_a was previously stored together with its solution s_a. Starting from the assumption that similar problems tend to have similar solutions, if s_a is a satisfactory solution to p_i, then the task is complete. However, if it is not satisfactory then s_a will still be subject to *reuse*, but in addition an attempt will be made to *revise* it into a suitable solution (s_i). If the change succeeds and the new solution is deemed to be of sufficient quality, then the CBR system will *retain* it in the case base for future retrieval.

The original 4RE model is deceptively simple. Retrieval requires an effective measure of similarity, and often uses an *index* to improve performance. Retrieval can be problematic if the case base is small, since there may not be a sufficiently similar case in the case base. If the case base is large, then performance issues may arise and there may be several similar cases to choose from. Reuse can be problematic if the query case and retrieved case are structured differently. Revision (otherwise known as *adaptation*) is almost always the most challenging step in CBR, since the mapping between problem space and solution space may be complex and poorly understood.[1] In some systems, adaptation is not automatic, and is left to the user; this is known as *null adaptation* [12]. Retention gives rise to decisions such as whether to retain all cases, or just the highest quality ones. Also, sometimes operations are performed to control the quality and efficiency of the case base, known as *case base maintenance*. These issues are described in detail in section 2.2.

Researchers have proposed amendments and extensions to the 4REs model, e.g. R^5 [13]. The R^5 model extends the traditional 4RE model with an initial *repartition* step. Repartitioning is based on the idea that the domain is divided into problems and solutions by a partition which "corresponds to a similarity relation". The aim is to control case base quality by using the similarity relation not just in retrieval, but also to construct the case base.

An entirely different definition of *repartition* is given in [14]. Here, an R^7model of CBR involving *case merging* is proposed, and repartition refers to the process in which a case is decomposed into constituent parts. Case merging is discussed in detail in section 2.2.3.3.

2.1.2 Foundations of CBR

Before the detail (of how CBR is performed) is discussed, it is important to examine CBR's origins. The theory of CBR has been influenced by the early systems, which were not developed solely as problem-solving tools, but also as an adjunct to research in cognitive science. This subsection summarises the major milestones in CBR research. For an amusing visual presentation of these milestones, see [15]. I have prepared a time-line of important CBR systems (figure 2.2), which is arranged by the date of the first significant publication. The shaded systems deal with case based design (see section 2.1.3.2).

2.1.2.1 Beginnings

Some researchers (e.g. [1, 9]) cite Wittgenstein [16] as being the earliest documented work that constitutes the theoretical foundation of CBR. However, the immediate roots of CBR were at Yale University in the 1970s with Schank's work on "dynamic memory" [7]. Schank formed a model of how he believed humans completed tasks by recalling events. In his model, the growth of the memory is triggered by failure. A *script* is the generalisation of a set of situations, whereas a *scene* is a specific instance of that script. A new situation may appear to occur within the context of a script; this creates an expectation.

[1] Adaptation is not required in some CBR systems, for example in many classification tasks.

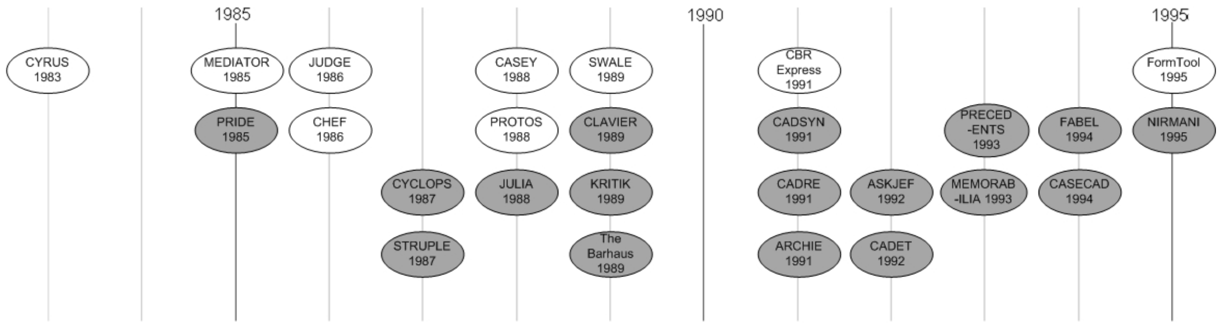


Figure 2.2: Time-line of Early CBR Systems

If the script does not encompass that situation, then there has been an *expectation failure*, and this new situation is stored within the dynamic memory. A *memory organisation packet (MOP)* stores a list of scenes, and the goal that they are directed towards. The dynamic memory is hierarchical, and a *thematic organisation packet* (TOP) is a more general concept than a MOP. Early CBR systems organised their case base in the way that reflected Schank's dynamic memory model. These early systems were meant to not only mimic human problem solving, but perhaps also provide a justification for his model.[2]

The first CBR system was **CYRUS**, by Kolodner [10]. The domain of CYRUS was events in the lives of the former US politicians Cyrus Vance and Edmund Muskie. The focus was on case structure and retrieval; there was no adaptation[3]. CYRUS was able to answer questions presented to it in natural language. If the system is presented with a new fact, it is able to incorporate it into its case base. Items were stored in "Episodic Memory Organisation Packets" (E-MOPs). An E-MOP has two parts: generalised information which describe its instances, and a conceptual tree which uses the differences between "episodes" (cases) to index them. MOPs were arranged into a hierarchy, with sub-MOPs for more specialised concepts. Indexes were chosen carefully by an algorithm which prioritised those that were likely to be "predictive". If features often occur together then one is said to be predictive of the other. The system was dynamic in that not only could new indices be created, but also new E-MOPs. It was the dynamic nature of the memory (the system was "self-organising") in CYRUS that made this truly ground-breaking work.

2.1.2.2 Proof of concept systems

Other researchers sought to build on the success of CYRUS, by providing evidence of CBR's capabilities. **MEDIATOR** [17] was designed to resolve disputes by suggesting compromises. It organised cases in a memory which was based on CYRUS. Problems were treated as a series of sub-goals, which begin with

[2] A caveat needs to be added to this discussion. Schank et al devised a model of how the human mind might work. Several early CBR systems used that model as the basis for their case memory, and those systems showed promising results. However, this does not constitute a proof that Schank's original model of the mind is correct.

[3] Adaptation was omitted from CYRUS because it was unnecessary. The purpose of the system was to retrieve facts about actual events in history. Adaptation would be required for activities such as speculating about events that are beyond the knowledge of the system (i.e. not in the case base) or inventing fictional events. However, this was outside the scope of CYRUS.

interpreting and classifying the problem itself. Each sub-goal was attempted using CBR, but if this failed then a series of heuristics, e.g. "dependency-directed backtracking" were used. It had built in policies, e.g. "divide equally", which it attempted to use to resolve situations; MEDIATOR was capable of providing a good textual explanation of its reasoning. A particularly interesting feature of MEDIATOR was how it dealt with failure; recovery from failure was treated as a problem to be solved. Failure was then remedied as per the main problem-solving strategies: first using CBR to search for previous failures, then re-interpreting the problem. Since a problem in MEDIATOR was divided into a series of sub-problems, it was able to use several previous cases to attempt to solve the main goal.

JUDGE [18] was a system that modelled judicial sentencing, in cases involving serious crime. In some ways criminal sentencing is a model task for a CBR system, as the stages in the CBR cycle can be clearly mapped to a judge referencing previous cases in a law library. During retrieval the judge finds the old case in the library, in reuse he cites it in his judgement, and in adaptation he alters the sentence to reflect the differing circumstances of the new and old cases. JUDGE modelled a simplified version of the circumstances, such as the offender's motives. An interesting feature was the "difference analysis procedures" which compare the retrieved case with the query case. JUDGE would iterate through the cases, looking for points of difference. It would then make a determination as to which of the two crimes was more severe. A noteworthy feature of JUDGE was its use of rules. If the query and retrieved case contained a substantial number of common features, a "sentencing rule" would be stored, with the common features used as an index. If a new case was presented and it matched a rule, then that rule could be invoked. JUDGE also had the capability to modify rules.

CHEF [19] was a case based planning system in the domain of Szechwan cooking. Its input consisted of a set of goals, and its output was a single plan (a recipe). CHEF maintained a map of features and how they mapped to particular problems; thus it was able to anticipate those problems. It contained "modification rules" for adaptation. If a plan failed, CHEF built a "causal explanation" and executed a "plan repair" process. The explanation was used to locate a thematic organisation packet (see above) which contained a strategy for dealing with the failures, e.g. reordering steps. The repaired plan was stored and indexed in such a way that such a failure could be avoided in future.

CASEY [20, 21] dealt with diagnosis in coronary heart disease. It was built on top of a *model-based reasoning (MBR)* (see section 2.3.2) program known as the Heart Failure program; this provided the output if the CBR did not work. Retrieval in CASEY worked by matching all the features of a case, using algorithms from the Heart Failure program; this often returned more than one case. Choosing the best case from those retrieved was referred to as *matching*; this considered only features judged to be important, which meant features that were relevant to similar cases in the case base. Cases were given a score which was derived from an algorithm that compared important features in the retrieved case and query case. The highest scoring case which constituted an acceptable match was selected, subject to a constraint that the retrieved case must not have a score that is too different from the score of the best match. A process known as *justification* determined if a match was acceptable; this used a series of domain-independent rules such as ruling out matches which contain values for features that could not possibly co-exist in the underlying causal model. CASEY's novelty lay in the integration of CBR and MBR; the combined system had greater capabilities than either of these alone.

PROTOS [22] was a general purpose "learning apprentice for heuristic classification tasks". It categorised objects by storing exemplars (of a category) with their features and an explanation of the features relevance to that category. A human teacher trained the system, which learned a "category structure". As per the work of Schank and others, a heuristic estimate of a particular category being relevant to a new case was known as a "reminding" (this is similar to SWALE, see below). Initially these remindings were gleaned from an explanation provided by the "teacher" (i.e. the controller of the system). New cases were integrated into the category structure, which may have involved merging cases judged by the teacher to be sufficiently similar. Merging resulted in features being generalised. PROTOS was trained with data from audiology. When an incorrect classification was produced, PROTOS chose new remindings and made different generalisations according to heuristics; these were retained if accepted by the teacher. PROTOS's novelty lay in the powerful general purpose nature of its capabilities.

SWALE [23, 24] was able to exercise creativity in its explanation of anomalous events. The name originated from a prize-winning racehorse, Swale, which died unexpectedly of an undetermined cause in 1984. Cases in SWALE were known as "explanation patterns" (XPs). Events were explained by SWALE by retrieving XPs, in a process referred to as *reminding*. Reminding was accomplished by traversing the hierarchy of XPs in the case base. Anomalies fell into categories, e.g. "premature event". If an existing event was already located in the hierarchy then the explanation that was stored with it was retrieved, and the process ended. However, if there was a failure to locate the event then a new explanation was attempted, by adapting or *tweaking* the existing explanations of other events. Sometimes tweaking used quite novel methods such as searching for explanations from folklore. The new explanation was then stored in the hierarchy. SWALE aimed to provide deep, rich adaptation, and to learn from adaptation failure. The outstanding features of SWALE were its ability to learn from failures, and the richness of its explanations.

These early systems proved that CBR was a promising methodology with a wide applicability. CBR was able to work in situations where there was a well-defined underlying model (CASEY), and also when there was not. CBR was shown to be robust, and able to handle failure well (MEDIATOR, CHEF). It was shown that CBR can be combined effectively with rules and heuristics. These early systems were also able to provide good explanations for their results, meaning CBR was not just a 'black box' approach, which can be important for gaining the trust of users.

2.1.2.3 CBR becomes a mature discipline

During the 1980s, CBR became an established sub-discipline within artificial intelligence, tested in principle with some academic systems and backed by a solid foundation of theory. During the 1990s this theory was extended, and commercial CBR systems were produced; three examples of such systems are presented here. These systems have been chosen because they are ground-breaking (The Bauhaus), they are widespread in the commercial world (CBR-Express) or because they have lead to substantial monetary savings (FormTool). Further examples of successful CBR systems are listed in [9].

According to Tierney [25], **The Bauhaus** [26] was the first commercial CBR system, developed by Inference Corporation. Component-based software development was automated by using a "frame-based description language" to implement retrieval of software components from a catalogue. The Bauhaus parsed textual specifications using the frame-based description language, then used a "subsumption network" to generate implementations using a breadth-first search. There was null adaptation, although a "data flow diagram editor" was provided to assist the user. The user would edit the flow graph, some contradiction checking was performed, then program code was generated. It is worth noting that since software is inevitably composed of more than one component, this is a form of *case-merging* (see section 2.2.3.3).

Inference Corporation were awarded several patents (e.g. [27]) for their enormously successful application known as **CBR-Express**, which originated from a project which automated software development at NASA [28]. CBR-Express is tailored towards the needs of helpdesks. Watson & Marir [9] make the point that as helpdesks often deal with faults, a disadvantage of a model-based reasoning system is that it may be easier to fix the fault in the product than to program the causal rules that the MBR would require. CBR-Express uses *nearest neighbour* matching in retrieval. It is able to take free-form text as an input, representing words as *trigrams* which are tolerant of spelling mistakes. A list of cases is retrieved, along with some questions; these are used to narrow down the list further. There is a threshold value for what constitutes an acceptable solution: if this is met then a case is retrieved. Otherwise, a more knowledgeable member of staff manually provides the solution to the case. CBR-Express is particularly noteworthy since it is domain independent and highly commercially successful. For example, it was deployed as a reservation system at American Airlines [28].

FormTool ([29], [30]) automates a colour matching system used at GE Plastics. The process involves dyeing plastic using different colourants, with the aim for the finished product to have a predetermined colour. The cost of the colourants must be minimised, and the accuracy of the finish colour must be maximised, no matter what the starting colour of the plastic was. A "fuzzy preference function" is used in retrieval. A physical model[4] is used in adaptation; unfortunately this only provides an approximation.

[4] The model used by FormTool is Kubelka-Munk theory.

The search space is so large that it is impractical to solve the problem alone using an unguided exhaustive search, which is why the CBR is required.

It is interesting to note that, before FormTool, GE Plastics used a semi-manual process which closely mirrored the principles of CBR. They stored previous successful colour matches in filing cabinets, which were manually indexed. These filing cabinets were the physical equivalent to a case base, or more exactly the physical predecessors of it. They manually retrieved a case, which consisted of a plastic chip which had been previously coloured. If this chip exactly matched the new requirement, the parameters that created it were re-used. Otherwise, the person's experience and a localised search (using software and human intuition) were used to adapt the old formulae to the new requirement. The adaptation was iterative, typically producing several sample batches of plastic in order to test the results. CBR has been able to provide more accurate matches in a shorter time period, requiring less batches, and as a result GE Plastics have reduced their expenditure by millions of dollars.

These three systems proved that CBR is more than a mere toy for academics. The emphasis moved away from attempting to mimic human memory and reasoning to practical systems that function well in commercial environments. CBR-Express showed how case based reasoning can be tolerant of *noise*, which is typically present in the "real world". All three systems could be used by people who were not computer scientists; and as FormTool has shown, sometimes the processes in CBR are not too different from the manual processes that it replaces (or supports).

2.1.3 Types of tasks performed by CBR systems

CBR systems can perform a wide variety of tasks, ranging from software design and dispute resolution. This subsection surveys the different types of tasks carried out by CBR systems, each of which has their different challenges, and a different subset of methods that are applicable.

Figure 2.3 shows a taxonomic hierarchy of tasks, adapted from [2] [5]. There are two major categories of tasks[6]:

- *Classification* tasks involve making a decision which results in the assignment of a category or label to an object, e.g. medical diagnosis.
- *Synthesis* tasks create an artefact, e.g. a plan which controls the movement of a robot in a factory. Creation of an artefact typically involves making very many difficult decisions.

This section compares synthesis and classification; there are several significant noteworthy differences between the two, e.g. classification tasks typically have many inputs and one output, whereas synthesis tasks tend to have few inputs and many outputs. Classification tasks can be subdivided further into

[5] A similar diagram to this also featured in [31].

[6] Marling et al [32] divide tasks into *interpretive* and *problem-solving*, which roughly translate into classification and synthesis respectively.

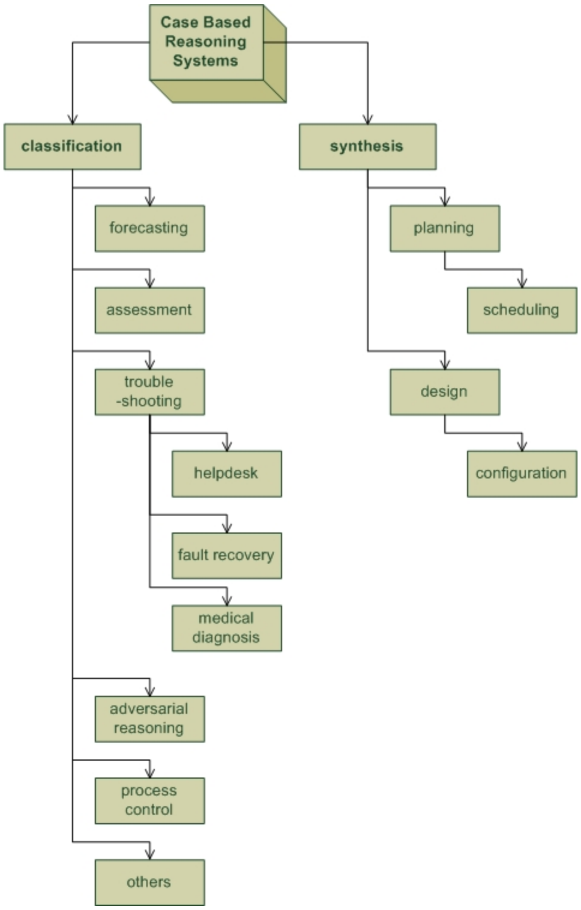


Figure 2.3: Taxonomic Hierarchy of CBR Systems, adapted from [2]

several categories (e.g. forecasting, assessment, and troubleshooting) and these are discussed below. Synthesis tasks are either planning tasks or design tasks, and these are covered in section 2.1.3.2.

2.1.3.1 CBR for Classification

When classification tasks are automated, a computer is being entrusted with a decision-making task which was traditionally being done by a human expert, e.g. a doctor. In artificial intelligence terms, this may be viewed as partitioning a problem space into clusters, which are given labels (e.g. "diseased" or "healthy"). In CBR systems which perform classification, the case base consists of a repository of previously labelled objects. A new object is presented to the system, and the best match for this is retrieved; the label for the object in the case base is then applied to the query case. There is typically no need for adaptation, since when a label is retrieved it can often be reused as it is.

Diagnosis is a typical classification task. The query case will consist of a set of symptoms without a diagnosis, and the case base will consist of previous cases which have both the symptoms and diagnosis recorded. Some systems have to be able to furnish a diagnosis even if some of the values are missing. Early examples are systems CASEY and PROTOS (see 2.1.2.2).

Systems for *help desks* are similar to diagnosis, however instead of the emphasis being on the cause of a problem, it is on the fix for that problem. CBR-Express (see subsection 2.1.2.3) is a well-known example of a helpdesk CBR system. One particular feature of helpdesk systems is that they often have to cope with textual input, which may contain errors. CBR Express deals with errors through the use of trigrams (three-letter words). The input text is split into a set of trigrams and this is used in the retrieval. For example, the trigrams for “PRINTING” would be {”PRI”, “RIN”, “INT”, “NTI”, “TIN”, “ING”}. If a word is misspelled, typically several of the trigrams are still correct and so retrieval is much more accurate than if the raw text were used.

An extensive discussion on the use of knowledge-based systems in helpdesks is presented in Dearden & Bridge [33]. There is a discussion of knowledge-based systems in general, and these are divided into:

- *domain-model based reasoning* (DMBR), which includes rule based reasoning and model based reasoning.

- *example-based reasoning* (EBR), which includes case based reasoning and others such as *analogical reasoning*.

Dearden & Bridge propose several arguments why EBR systems are more appropriate for helpdesks than DMBR systems. For example, they make the point that DMBR systems are often particularly complex, and substantial effort is often required to construct and maintain them.

Fault recovery involves both diagnosis and recovery. Althoff et al [2] say that, since there are often existing procedures in place to recover from faults, these type of tasks can often benefit from a hybrid system involving both CBR and *model based reasoning* or *rule based reasoning*. Such hybrid systems can be very powerful since the different techniques can complement each other and, to some extent, make up for each others shortcomings (see section 2.3). For example, rules may be useful at the beginning, since sparsely populated case bases tend not to perform well as there is typically too large a gap between the retrieved case and query case. When the case base has more cases in it, the CBR could then prove invaluable by reasoning in situations that are not covered by the rules.

In *forecasting* and *prediction*, the case base consists of a repository of outcomes, which are stored together with features that are thought to be predictive of that outcome. Temporal data is particularly prevalent in these types of task, and some systems work with aggregations of the data [2]. In this type of problem, it is particularly important that the user is confident in the output. CBR systems are often capable of providing an explanation of their output, which is especially advantageous when compared to other techniques which are more like a “black box”, e.g. neural networks.

Althoff et al [2] refer to tasks such as *credit scoring* as *assessment*. These tasks are different from forecasting since they attempt to objectively evaluate the present situation, without attempting to predict the future.

Adversarial reasoning is where a human process which has two competing parties is modelled, e.g.

legal disputes. For example, in an attempt to exonerate a defendant in a criminal trial, a system may encounter previous similar cases with verdicts of both 'guilty' and 'not guilty'. Retrieved cases are optimal if they carry the label or class which the user requires; other cases carry a conflicting label. These sort of systems have a "point-counterpoint reasoning mechanism" [2], which aims to win an argument by anticipating the opponent. A system will typically look for similarities between the query case and optimal case, concentrating on features with values that are very different from those found in the sub-optimal cases. MEDIATOR (see section 2.1.2.2) is a classic example of CBR in adversarial reasoning.

In [33], the argument is advanced that, when a helpdesk system proposes a fix, this is a synthesis task (referred to by them as "construction"). The authors also refer to several different hierarchies of types of tasks, suggesting that the one in figure 2.3 is not definitive; however it is adequate in my opinion. Classification is a term used generally in the literature of machine learning, and it generally refers to a requirement to the decision as to which label or category an artefact should bear. Since these types of tasks are arguably easier than synthesis tasks, it comes as no surprise that systems to automate classification are particularly common. However, it should be noted that there are other ways of classifying data (e.g. neural networks, decision trees, or support vector machines). These other methodologies are not necessarily applicable to synthesis tasks, which are considered in the next section.

2.1.3.2 CBR for Synthesis

In synthesis tasks, CBR uses a previous artefact as a template for creating a new one [2]. Adaptation is typically extensive in synthesis tasks, whereas in classification tasks it is often absent. Furthermore, as Althoff et al point out, adaptation is often highly dependent on the application domain. Synthesis tasks tend to be more difficult to automate than classification tasks, irrespective of the nature of the automation (CBR, rules, model-based reasoning, etc.):

> "As regards representation, indexing, retrieval and adaptation, creating planning/synthesis problem solvers is a higher order of magnitude of difficulty than building classification-oriented problem solvers" - Althoff et al [2].

Planning is the construction of a sequence of steps that achieve a defined goal, given a specified initial state. Planning has been an extensively researched area of artificial intelligence for many years (e.g. [34]). Automated planning systems are often used in project management, and manufacturing (Nau, [35]). Nau makes the distinction between planning, where the actions (that will be performed) are undetermined, and *scheduling*, where the actions are predetermined but their ordering is yet to be decided. Successful automated planners are normally domain dependent; domain independent systems are very hard to develop since planning problems can be NP-complete, or even worse, they can be *undecidable*. Traditionally, a variety of heuristic algorithms (for example, Hierarchical Task Network planning) are employed to traverse the search space, which can be extremely large.

Case based planning works by retrieving a previous plan from the repository, and adapting it to a new requirement. One issue discussed in [2] is *granularity*: a case can be a whole plan, or just part of a plan. The issues around granularity are similar to those encountered in case-merging (see 2.2.3.3).

- If *a case is a whole plan*, then the representation is more complex, indexing of cases is harder, and adaptation is more difficult (since there are fewer cases and the cases are more complex). However, if the retrieved case happens to be very similar to the query case, then there is much less work to do than if the cases were smaller.

- If *a case is only a part of the plan*, case representation, indexing and adaptation is easier, due to reduced complexity. However, retrieval is more complex since it requires a more specialist search operation: the retrieved parts have to be compatible with each other. Systems can also include parts in the solution that are not from a previous case, but have been derived from first principles [36].

CHEF (see section 2.1.2.2) is often cited as an example of an early case based planning system. More recently Darmok, a system that focused on the domain of real-time strategy games, has been presented [37]. A key issue in case based reasoning is how to initially populate the case base (this is discussed briefly in section 2.2.4). The authors show how this issue can be resolved by enabling plans to be automatically learned from human demonstrations. Darmok generates a goal matrix which links a set of goals to a set of plans, removes unnecessary actions through a dependency graph, then converts plans into hierarchies in order to check if a plan is in fact a sub-plan of another. It uses an algorithm known as Adversarial case based Planner (ACBP), which works by retrieving whole plans from the case base, then using transformational adaptation (see section 2.2.3.1).

Nebel and Koehler [38] compared the *worst-case complexity* of plan generation (by exploring the search space) and plan re-use. Their conclusion was that it is not possible to prove that reuse is more efficient than generation (by the means discussed above). The authors conceded that reuse can be faster in some situations, but said that retrieval is a bottleneck in plan reuse. However, Muñoz-Avila and Cox [36] question the assumptions of Nebel and Koehler, about the fact that the new solution is obtained by minimal modification of the retrieved plan. They point out that *derivational replay* (see section 2.2.1.2) violates this assumption, since it involves quite extensive adaptation. This debate could be generalised to CBR tasks other than planning, where *case merging* (see section 2.2.3.3) is used. However, efficiency is not the only factor effecting the choice of problem solving methodology: quality is also often an issue.

Design involves bringing together component parts to create an artefact, which conforms to a *specification*. Examples of design are ubiquitous since most man-made objects have been designed, and design is an activity of paramount importance in modern society. A key difference between design and

Factor	*Planning*	*Design*
Name of simpler type of task	scheduling	configuration
Involves temporal data	yes	rarely
Involves a spatial element	sometimes	often
Involves artistic/aesthetic factors	rarely	often
Can be automated using rules and/or heuristics	yes	sometimes
Decomposable into sub-tasks	yes (temporally)	yes (spatially)
Involves constraints	yes	yes
May be intractable, if constraints are too tight	yes	yes
Amenable to domain-specific solutions	yes	yes
Unambiguous and consistent specification?	usually	sometimes
Prevalence of the task in industry	widespread	widespread

Table 2.1: Differences and similarities between design and planning

planning is that in most planning problems discussed in the CBR literature, the objective is unambiguous. However, in design problems, the objective (the specification) is not always clear; it can be inaccurate, incomplete, and inconsistent [39]. For example, the designer might be tasked with making an object as aesthetically pleasing as possible, with the maximum functionality, but at minimum manufacturing cost. I have highlighted some common similarities and differences between planning tasks and design tasks in Table 2.1.

Design is often a complex task; however, paradoxically, it can be repetitive, and many new designs are variations on existing designs. This makes design a good candidate for case based reasoning, and indeed *case based design* is an active research topic in the literature.

The discussion of the different types of tasks that CBR is capable of automating shows how those concepts are applicable in a wide variety of application areas. If any two CBR systems were picked at random, although they would both work according to the same basic principles, they may look very different and concern different types of tasks. These wide differences have to be borne in mind when CBR systems are constructed.

2.2 Challenges in constructing a CBR system

This section discusses the choices to be made during the design and construction of CBR systems. Each of the stages in the 4RE cycle (depicted in figure 2.1) entail design choices. As with other AI methodologies, the efficacy of CBR is often greatly dependent upon the right decisions being made.

Typically, the first decision to be made is how the cases are represented or structured, and this is discussed in detail below. Closely related to the representation are the decisions about case indexing, often crucial since it can dramatically improve the efficiency of retrieval. However, the most important factor in retrieval is usually the similarity function; there are a wide choice of types of functions ranging from simple weighted sum algorithms to sophisticated graph-theoretic ones.

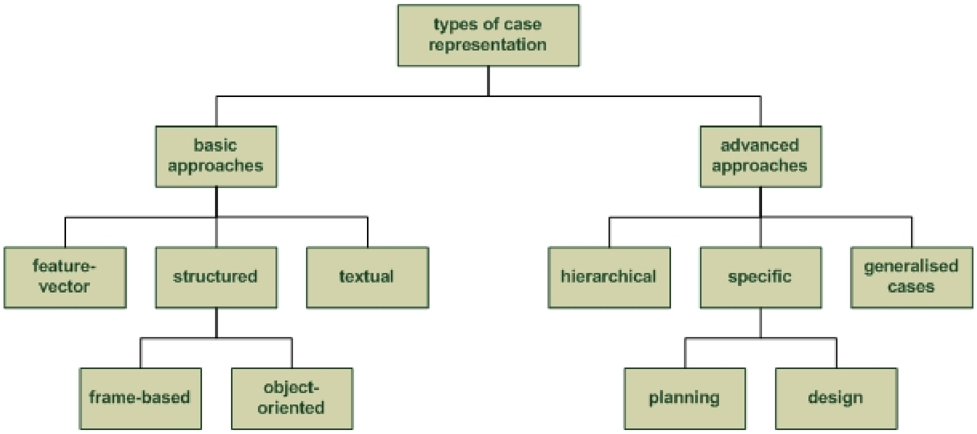


Figure 2.4: Types of case representation, as discussed in [3]

Adaptation is typically the most challenging part of CBR; it is an inherently complex task and it is often omitted, either because it is not required or it is left to a human user. However, some systems do successfully automate adaptation, and the different ways of accomplishing this will be discussed in detail. Retention is important as it can affect the efficiency of a CBR system, as well as its quality. Sometimes, in addition to the "online" decision about whether to retain a case, separate case base maintenance activities are performed. These activities are aimed at maintaining a level of quality in the case base, and there are both automated and manual methods of accomplishing this.

Finally, CBR shells are discussed. These purportedly reduce the time taken to implement a CBR system, by providing a generic framework which can be augmented with specific functionality if required.

2.2.1 Case representation

A useful discussion of what the content of a case should be is given in [40]. Kolodner divides a case into three parts: a description of the *problem*, the *solution*, and the *outcome*. The outcome is a description of the effects of applying the solution. Some CBR systems store a *trace* of how the solution was applied. However, it should be noted that in many CBR systems, the outcome will not be stored, because it is an obvious consequence of the solution, e.g. in diagnosis tasks. Therefore, the pairing of problem and solution constitutes the minimum composition of a case.

Watson describes numerous ways in which storage of problems and their solutions can be implemented, for example in a database [41]. As Watson argues, CBR is a methodology and not a technology, and it can be implemented in a variety of ways. Therefore, this section focuses not on the implementation details, but instead on the types of structures that are used to organise the cases.

Different approaches to case representation are discussed in Bergmann et al [3], and these are summarised in figure 2.4. The types of representation are divided into basic approaches and advanced approaches, although this is somewhat arbitrary since object-oriented representations could be considered an advanced approach.

2.2.1.1 Basic approaches

The simplest form of case representation is a so-called *feature vector*. Feature vector representations[7] consist of a set of features, each of which has an associated value. A feature can typically be described by a string name, and a value is often numeric. This type of representation is ubiquitous in machine learning, and is probably the commonest in CBR, due to its simplicity.

PROTOS ([22], see section 2.1.2.2) used feature vector representations within a *category and exemplar model*. The case base was a network of categories, cases (referred to as exemplars) and index pointers. There were three sorts [1] of index pointers :

- *remindings*: features were linked to either categories or cases
- *exemplar links*: cases were linked to their associated categories
- *difference links*: cases were linked to other cases that were very similar.

New cases were classified by considering their similarity to the existing exemplars, this is done by a heuristic known as "knowledge-based pattern matching" which took into account factors such as the prototypicality of the exemplar. The features simply consisted of just a name and a value; however, they were incorporated into the complex hierarchical network described above. This illustrates the point that considerable variation is possible within the categories in figure 2.4.

Bergmann et al talk about two types of structured case base representation: *frame-based* and object oriented approaches [3]. Minsky [42] defines a frame as a "a data structure for representing a stereotyped situation, like ... going to a child's birthday party". Herein lies the obvious relevance: a case can be thought of as such a stereotyped situation.

Description logic (a formal knowledge representation language) has been used to partially formalise frame-based approaches [3]. Use of description logic in CBR is discussed in Salotti and Ventos [43]. The case base was arranged in a taxonomy, and retrieval used two concepts: similarity and dissimilarity. The similarity between two cases was determined by finding the most similar concept in the taxonomy that subsumed those two cases (the "least common subsumer"). Dissimilarity worked by considering the relative complement[8] of the sets of attributes of the two cases.

As with frame-based representations, *object-oriented* (OO) representations can involve a hierarchy. OO approaches use features of an object-oriented language such as inheritance (or *is-a* relationships), and composition (for *has-a* relationships). Göker et al [8] described HOMER, which had an object-oriented case representation. HOMER was an in-house helpdesk system for automotive engineers at Daimler-Benz. The developers eschewed a "flat" feature-vector representation as there would be hundreds of attributes

[7] Otherwise known as "attribute-value" representations [8].

[8] Given two sets A and B, the relative complement of A in B is the set of elements that are in B, but not in A.

and determining which of these were relevant for the similarity function was thought to be an onerous knowledge engineering task. Also, a judgement was made that in order to solve "difficult" problems, the domain would have to be accurately modelled. The case was modelled in three parts: failure, symptoms and solution. The parts were related to each other in the object-oriented model, for example a failure could result in several symptoms.

New cases in HOMER were initially stored in a buffer, until they were validated by a human editor. The editors were given the responsibility of maintaining consistency and avoiding redundancy in the case base; they were allowed to tweak cases by modifying the values in them, but were not allowed to add or remove attributes. The use of an editor meant that the case base was quite small, and contained *prototypical* cases. HOMER could be operated in two modes: "user-driven mode" enabled an experienced operator to build direct queries against the case base. "System driven mode" was designed for less skilled operators; it generated questions for the operator to ask a user, based on selecting attributes with the highest *information gain*. The answers to these questions were used to retrieve suitable cases from the case base; if none were found then the user was allowed to enter their own solution, thus creating a new case which was then scheduled for review by an editor.

Göker et al stated that they chose CBR over rule-based systems, as they thought the latter would be difficult to implement. Nevertheless the development of HOMER apparently represented an unexpectedly difficult knowledge-acquisition task. This illustrates the trade-off between the time-consuming operation of building a complex model which accurately models the domain, and using a crude representation which is easier to construct but less powerful and flexible. Also, it shows that when developing a CBR system, one must consider not just the task and domain, but also the characteristics of the users. Highly skilled users are able to perform some operations manually that would otherwise be automated, such as retrieval (by constructing their own queries), and case based maintenance (by validating the cases). The level of automation that is appropriate may be dependent on the skill level of the users.

If the information in a case is purely or mostly *textual* then this is typically decomposed into *information entities* [3]. Information entities are particularly relevant words or phrases that can be placed (along with their associated cases) in a directed graph known as a *case retrieval net* (CRN). Retrieval in a CRN works by first "activating" information entities which are contained in the problem; this activation is then propagated to similar nodes in the CRN, in a mechanism that is analogous to activation propagation in neural networks.

If the inputs to the CBR system consist primarily of text, then a textual representation is the obvious choice. An alternative is to construct a parser which generates a feature-vector representation from the text; this has the advantage that it could incorporate domain-specific features, however its major disadvantage would be the effort required in constructing such a parser. It could also be argued that leaving the representation in its natural state (text) avoids information loss, since a feature vector representation

Ref	*Domain*	*Comments*
[44]	design of control software in steel mills	A design is stored in **Déjà Vu** as a hierarchy of cases: "abstract cases" describe approximate designs whereas "concrete cases" are fully implemented designs
[45]	automobile troubleshooting	A case in **CELIA** is a series of "snippets", each of which has a subgoal; snippets are linked together[9].
[46]	design of hydro-mechanical devices	Directed graphs are used in **CADET** to represent the behaviour of devices. New solutions are found by using two rules (chain rule and total inference rule) or their inverses.
[47]	control of humanoid robot soccer players	Behaviours are described on a 4-level hierarchy, the topmost of which is to decide the robot's role as an attacker.
[48]	resolution of complex conflicts in air traffic control	The hierarchy had 2 levels: one contains the problem as a whole, and the other has each pair of aircraft that are in conflict.
[49]	a simple route finding task	They compared various algorithms; some returned "abstract solutions", e.g. movement on an 8x8 grid was approximated to a 4x4 grid. They found that reusing these "abstract solutions" gave better performance than reusing only "concrete" ones.
[50]	balanced scorecards: a corporate management decision tool	Each feature had three levels of weights corresponding to different aspects of the scorecard; they used genetic algorithms to learn the weights.
[14]	knitwear design	**SEACOP** is the subject of this thesis.

Table 2.2: CBR systems using hierarchical representations

is going to be (at best) a summary of the text.

2.2.1.2 Advanced approaches

Hierarchical representations [3] possibly involve multiple levels of detail and different vocabularies. I have listed examples of several CBR systems which use hierarchical representations in table 2.2. Many early CBR systems such as CYRUS (see section 2.1.2.1) used a hierarchical structure that was based on models of the human mind.

In Smyth et al [44], the term hierarchical case based reasoning is defined as involving the retrieval of multiple cases, at different levels of abstraction. Many hierarchical systems involve case-merging, which is discussed in section 2.2.3.3. As well as reducing the sophistication of the adaptation, Smyth et al also argue that hierarchical CBR can reduce redundancy by sharing cases amongst hierarchies, thus making storage more efficient. However it is worth noting that this will not always be possible or practical; the lowest level in the hierarchy may contain data that is unlikely to be reused due to its complexity or uniqueness. Smyth et al also argue that the abstract cases (or parts of cases) can be used to facilitate indexing of the more detailed parts, this is a significant advantage since it will reduce the number of index entries, thus making retrieval more efficient.

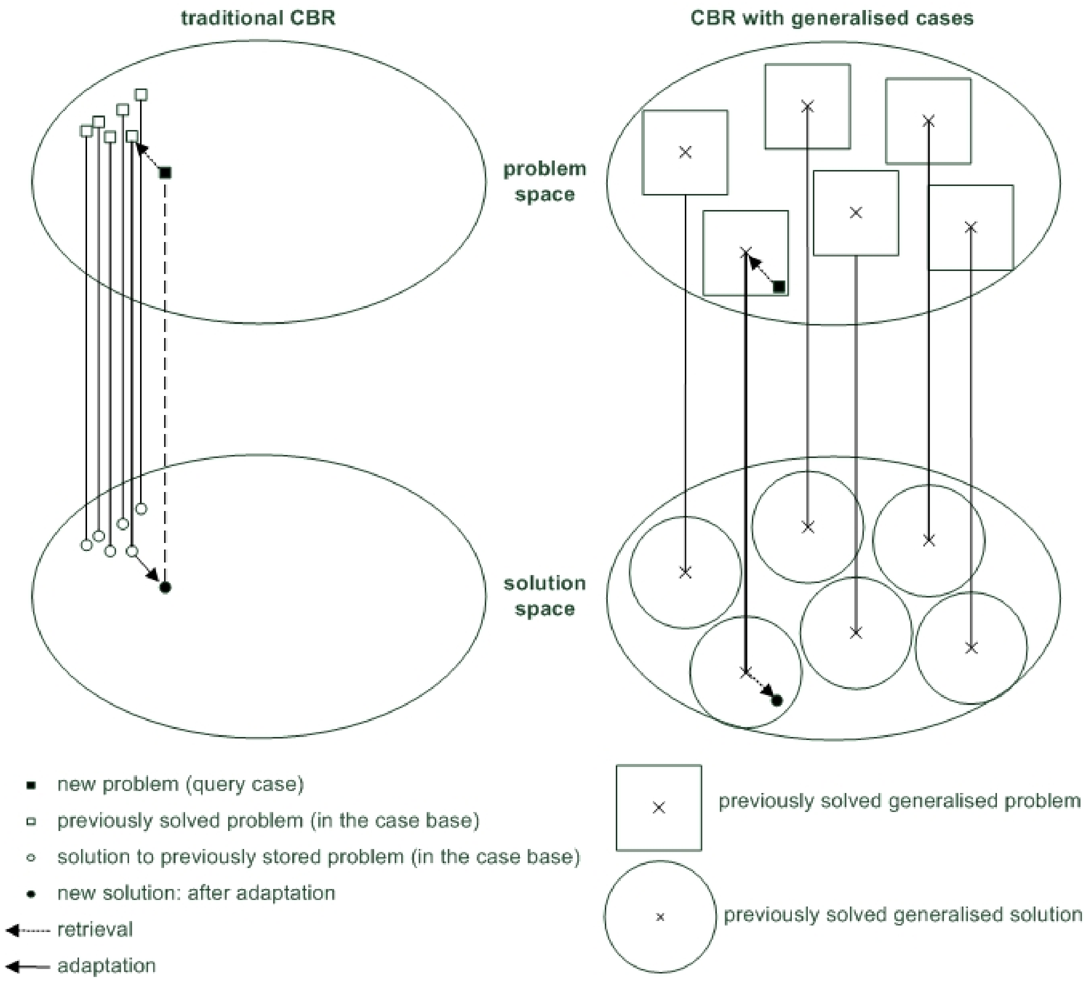

traditional CBR
CBR with generalised cases
problem space
solution space
new problem (query case)
previously solved problem (in the case base)
solution to previously stored problem (in the case base)
new solution: after adaptation
retrieval
adaptation
previously solved generalised problem
previously solved generalised solution

Figure 2.5: Traditional CBR versus Generalised Cases (adapted from [4])

Hierarchical representations are often thought of as being complex; however, the domains that they are used for (e.g. design, or planning) are often inherently complex. If the problem domain is not naturally hierarchical, then the abstract parts of the hierarchy need to be artificially constructed [44].

Bergmann et al [3] discuss *generalised cases*. Whereas a case normally relates to a single experience, scenario or problem, a generalised case can provide solutions to a range of problems. This is implemented by introducing variables into the case representation, which may have constraints applied to them.

Figure 2.5 (adapted from [4]) shows a comparison of the principles behind traditional CBR, with CBR using generalised cases. On the left, the two ellipses depict retrieval and adaptation in traditional CBR. *Only a few cases are shown* but as this analogy shows, sufficient coverage of the case base is required to make it work. On the right, the use of generalised cases is illustrated; similar processes of retrieval and adaptation are present, but far fewer cases are required since each case covers a range of the problem space rather than a single point.

A CBR system with a small case base may still perform well if it employs generalised cases [14]. It is also claimed [51] that generalised cases can ease adaptation by “filling the gap” in between cases (which are specific) and rules (which are general).

An alternative to generalised cases is to use *interpolation* [52]. In interpolation a set of cases which are close to the required solution are retrieved. The values in these cases are then used to derive the solution. Interpolation may be a useful technique if a good interpolation function or method is available, however it is unlikely to be suitable for complex cases where there are constraints or interdependencies between features, e.g. those used in design tasks.

In [3], several examples were given of very *specific case representations for design* tasks. For example [53] describes DRAMA, a system for aerospace design that utilises *concept mapping*. Concept mapping is defined as a 2-dimensional visual representation of concepts which shows their interrelationships. Often, the vertical axis on the map is used for different layers in a hierarchy. Concept maps were incorporated into the case representation of DRAMA so that the user could manually edit design cases. There is also a textual element to the maps, and this can be used to document the decisions that users have made during the design process. This is an example of the sort of extra functionality that is required when a CBR system acts as an assistant to a user, rather than fully automating a process. Experts are often required to document their work, and it is natural that this is recorded with the work that the documentation relates to.

Finally, as per figure 2.4, there are *specific case representations for planning* tasks. A notable example is *derivational replay,* in which the means by which an artefact was created are stored, reused, and adapted. In [54], derivational replay was implemented in a domain independent architecture, in which goals were divided into subgoals and solved recursively. The case base was changed in response to feedback from a problem solving module, which used a backward-chaining search. The steps taken by the problem solver, the justifications for those steps, and even failures are recorded. Since the system was domain-independent, the steps taken to solve a problem in one context were able to be replayed in another context.

I believe that derivational replay is most likely to be successful in scenarios where automated planning systems are successful: i.e. where the problem can be completely represented in some formal language or calculus, there is a goal that can be divided into subgoals, and a large search space makes it a non-trivial problem. The trade-offs between retrieval and search can be managed well, as the system is able to avoid repeating past failures (since those failures are recorded). In my opinion, it is unlikely to be successful in scenarios where the CBR acts as an assistant for a human user, since humans often solve complex tasks by means which are inconsistent, but that are acceptable.

2.2.1.3 Discussion

The previous sections have shown that there are several types of case representation, and these can vary widely in their complexity, from simple feature-vector approaches to complex hierarchies. Even within one type of representation, there is considerable variation; for example there is no universal model of a hierarchical case representation; some systems have hierarchies of cases (e.g. abstract and concrete cases), others have hierarchies within a case (and they can differ in the number of levels in the hierarchy).

Sometimes the data which constitutes the inputs to the CBR process will be structured in such a way as to make the choice of representation obvious, e.g. if it is textual. If the data is complex, then the developer must choose whether to accurately model this in the case representation, or whether to construct a simpler object (e.g. feature-vector) which is added to the complex data and does the work involved in the CBR. A complex representation will be more flexible and can use more domain-specific similarity measures and adaptation operators. A simple representation may be easier to index and easier to retrieve. However, if the data itself is not simple, then the disadvantages to this approach are the effort required to construct the the algorithm that simplifies it, and the loss of information.

Some problem domains are more amenable to certain types of representation than others, e.g. derivational replay for planning, feature-vector for a simple helpdesk application, or hierarchies in design. Also one must consider the nature of the task: is the system an assistant to a user, or does it totally automate a process? If it is the latter, then the user will have other requirements such as the ability to annotate their decisions. If a CBR system has additional functionality, e.g. if it is also a CAD program, then this may influence the choice of representation as the data is being used for more than one purpose.

Finally, and most importantly, the representation must support the choice of similarity function, and adaptation operators (if applicable). For example, feature-vector representations tend to support weighted sum type algorithms very well. Similarity and adaptation are discussed in more detail in sections 2.2.2.2 and 2.2.3 respectively.

2.2.2 Retrieval

In the retrieval step of CBR, the system finds a case from the case base which is sufficiently (or maximally) similar to the query case. Sometimes, retrieval is the most important step in the CBR cycle; this is particularly true if there is null adaptation. Even if there is adaptation, retrieval is arguably as important as adaptation: if retrieval is sub-optimal it may select a case (or cases) which are not sufficiently similar to the query case, so that adaptation becomes difficult or impossible.

If there are only a few cases, the the retrieval algorithm may simply iterate through the case base, applying the *similarity function* (see section 2.2.2.2) to each case. This is known as *linear retrieval* [2], and might apply if generalised cases are being used, or perhaps if the case base is being kept very small with prototypical cases. However, often case bases are too large to be searched in their entirety, and one solution to this is to use indexing; this is discussed in section 2.2.2.1.

The concept of similarity (between cases) is central to retrieval. Different ways of quantifying similarity are discussed in section 2.2.2.2. The techniques are categorised by whether or not they require training, and whether they are *knowledge intensive* or *knowledge light*. The latter distinction is important: knowledge light approaches involve little or no domain knowledge, whereas knowledge intensive approaches specifically incorporate domain knowledge.

The similarity measure will be used by the retrieval algorithm in order to select cases from the case base. A detailed discussion of retrieval algorithms is presented in section 2.2.2.3.

2.2.2.1 Case indexing

Case indexing can be described as the capability to exclude a portion of the case base from the retrieval process [55]. The objective of indexing is to avoid the *utility problem*, which is is pervasive in artificial intelligence. In methodologies such as neural networks or support vector machines, the utility problem manifests itself when excessive training has occurred. This is due to *over-fitting*: the mapping (between inputs and outputs) becomes unnecessarily complex, and fits the training data too closely. When data is supplied which is outside the boundaries of the training data, it is often misclassified.

In CBR a large case base does not necessarily result in a drop in competence, but is associated with a reduction in efficiency [56]. As the coverage of the problem space becomes denser, the mean distance between retrieved case and query case decreases, and hence the mean effort required for adaptation decreases.[10] This reduction in adaptation effort becomes progressively smaller as more cases are added. However, the effort required for retrieval increases as the case base becomes denser, simply because there are more cases in the case base that require comparison with the query case. Eventually a point is reached whereby the advantages of adding cases (better coverage) are outweighed by the disadvantages (inefficient retrieval).

The utility problem is an issue if it causes retrieval to be unacceptably slow. The efficiency of retrieval will be influenced by several factors including the complexity of the retrieval algorithm, the size of the case base, and the computing power that is used. So, to alleviate the utility problem one must increase the processing power assigned to the task, perform case base maintenance to limit the size of the case base (see section 2.2.4), or use indexing.

Kolodner [57] divides the problem of indexing into two sub-problems:

- *Labelling cases*: the data or metadata that labels a case must allow the similarity function to quickly decide whether or not that case is suitable for retrieval.

- *Organising the case base* so that the case base can be searched efficiently.

The problem of indexing is not a well-bounded one: case representation, case indexing and the other aspects of retrieval are inter-related. A case must be labelled with an understanding of its representation, and in such a way that it supports the similarity function. The retrieval algorithm must take into account how the case base is organised. Case representation is not synonymous with organising the case base,

[10] Provided the assumption that *similar problems require similar solutions* is correct

since the former is focused on the structure of an individual case, whereas the latter is concerned with the structure of the case base as a whole. However, sometimes these two are related, for example if cases are arranged in a network with pointers to other cases, then case structure and case base structure are intertwined.

Kolodner[57] describes the desirable features of indices. Indexing should be predictive, useful and achieve the correct level of abstraction. *Predictive* features are those that are relevant to the "outcome" or success of the solution in the case. However, in some CBR systems, the outcome might not be explicitly stored. Perhaps a more useful idea is that features should be chosen on the basis that they provide a good contribution to the mapping between problem and solution. For example, in a medical system predictive features might be those symptoms that best discriminate between the different diagnoses.

Kolodner [57] describes *abstract* indices as being widely applicable. If the features of a case are hierarchical, then more general versions of those features would have that wider applicability. For example, in the culinary domain, a dish could be described as containing fruit rather than apples. If the features were numerical values then fuzzy sets could be used to make these more general. Kolodner also says that indices that are too general may be unusable. In early CBR systems where cases were arranged with concepts in a hierarchical network, retrieval based on generalities increased the complexity. In more modern CBR systems which use feature-vector representations, for example it is certainly true that making an index too general might weaken the similarity algorithm. It could reduce the algorithm's ability to differentiate between similar cases, or it might simply result in inefficient indexing, i.e. too many cases being selected.

Kolodner states that indices should be *useful*. Useful indices are described as "those that label a case as being able to give guidance about the decisions that reasoner deals with" [57]. For example, many CBR systems are capable of learning from failure. If including a feature as part of an index is likely to help the CBR system avoid failure, perhaps by simply not retrieving that case, then this could be valuable. However, it is worth noting that CBR systems can reason using data in the whole case, not just the index data.

The previous discussion (from [57]) is suggestive of indices being chosen manually, by a domain expert. However, much work has been done on the automatic creation of *decision trees* for indexing. Two classical decision tree algorithms are k-d and ID3 [2]. A *k-d tree* is a type of binary search tree, in which the vertices constitute a subset of the case base (or indeed the whole case base, in the case of the root vertex). Each non-leaf vertex has two child vertices which partition its cases into disjoint sets. A well constructed k-d tree will be balanced, since its height is then minimised and the minimum number of nodes are traversed, thus making it more efficient. The *ID3* algorithm was introduced by Quinlan [58]. It uses a greedy algorithm to build a decision tree based on a heuristic known as *information gain* which picks the most discriminatory of attributes. The trees generated are not necessarily binary, but tend to be well balanced [2].

There has been extensive research which has improved on the classical algorithms. For example, Galushka and Patterson [55] describe D-HS^{E}, which is a domain-independent algorithm that can be used to automatically construct indexes for a given case base. In D-HS^{E}, attributes are split into intervals; if the data was discrete then it is left as it is. However, if the data is continuous then it was divided into intervals using an "entropy-based discretisation approach", which chose the intervals with the aim of achieving a high information gain. They tested their technique using a standard dataset and replaced missing values using averages.

Decision trees are simple to understand, can be automated with little or no manual intervention, and being domain-independent they are generally applicable. They are also efficient: a balanced binary tree with n nodes requires at most $\log_2 n$ decisions.

Decision trees also have a number of significant disadvantages. They cannot tolerate missing values, so these are typically substituted with the mean or modal value of the other data. This is obviously undesirable, but the only alternative is to remove the feature (which can have missing values) from the tree. Discretising continuous attributes will always result in loss of information. Trees are built using heuristics and work well for some data sets but not for others [55]. One of the claimed advantages of CBR systems is that they cope well with change, however this may not be so if a decision tree is used. If the character of the case base changed, unexpected degradation in the performance of the indexing tree could adversely affect the output of the CBR system.

When decision trees are implemented in a CBR system, they typically partition the cases repeatedly until the number of cases reaches a specified threshold [2]. The cases that remain are then tested against the query case with the similarity function. With a threshold value of 1, the decision tree becomes the whole retrieval mechanism; at the other extreme, if the value is very high then the decision tree will be very shallow. If the threshold is a fixed value, then the decision tree will deepen as the case base grows; the time taken for retrieval will not increase appreciably but more and more of the reasoning will be performed by the decision tree. By carefully setting the threshold it may be possible to achieve the optimum combination of efficiency (due to the decision tree), and accuracy (with a well-chosen similarity function).

2.2.2.2 Similarity

The key assumption of CBR is that *similar problems require similar solutions* (see section 2.4.1). CBR expects some degree of correlation between similarity[11] and adaptation distance, as illustrated in figure 2.5. Therefore, it is always important that there be an effective similarity measure for CBR to work.

[11] In this section, I have used both the terms *similarity* and *distance*. For our purposes, the terms are assumed to be the inverse of each other, hence the greater the similarity between two objects, the lesser the distance.

distance	*known as*	*formula*
L_1	Manhattan or city block	$\sum_{r=1}^{n} \lvert f_r(x_1) - f_r(x_2) \rvert$
L_2	Euclidean	$\sqrt{\sum_{r=1}^{n} (f_r(x_1) - f_r(x_2))^2}$
L_{P}	Minkowski	$\left(\sum_{r=1}^{n} (f_r(x_1) - f_r(x_2))^P\right)^{\frac{1}{P}}$
L_∞	Chebyshev	$\max \lvert f_r(x_1) - f_r(x_2) \rvert$

Table 2.3: Minkowski distance formulae

This subsection discusses the types of measures that can be used for similarity, ranging from a simple weighted sum approach to more complex measures, such as those based on information theory. A useful taxonomy of similarity measures is given by Cunningham [59]. The measures were grouped into four categories:

- direct metrics
- transformation based
- information-theoretic
- emergent

Direct metrics are the most established, and probably the simplest of the types of measure. Direct metrics are used to find the *nearest neighbour* in the problem space, or in general several near neighbours (known as k-NN). These metrics are applied to feature-vector representations, that constitute the simplest way to structure a case (see section 2.2.1.1). The similarity measure of two cases will have two components: *local similarity*, which measures the similarity between the values of an attribute in two different cases, and *global similarity*, which is an aggregation of the local similarities that applies to the two cases as a whole.

The simplest direct metric is *equality similarity*, which involves simply counting the pairs of attributes which are equal to each other. However, equality similarity is rarely appropriate since it only takes into account the fact that attributes are different, not *how* different they are. It is likely to be more applicable to symbolic data than continuous numeric data.

A common measure of distance involves summing the differences between numeric values, known as the *Manhattan distance*. The Manhattan distance is a special case of the *Minkowski* distance. I have given the formulae for the Minkowski differences in table 2.3, for two cases cases x_1 and x_2 which are described by a feature vector $<f_1, f_2....f_n>$. $f_r(x_1)$ refers to the value of feature f_r in case x_1. As the value of P increases, more emphasis is placed on the pairs of values which are most different from each other, until eventually in the *Chebyshev* distance, only the maximal difference is relevant.

Euclidean distance is a common variant of Minkowski distance, with P=2. It has a reported tendency [60] to perform poorly when a lot of values are zero, in which case another alternative is the *cosine angle distance*, which is defined as:

$$cosine\left(x_1.x_2\right)=\frac{\sum_{r=1}^{n} f_r(x_1) f_r(x_2)}{\sqrt{\sum_{r=1}^{n}(f_r(x_1))^2}\sqrt{\sum_{r=1}^{n}(f_r(x_2))^2}}$$

Cosine angle distance is described in [61] as providing similar results to Euclidean distance, but having the advantage that it is automatically *normalised*. For some applications, such as finding the maximum similarity, normalisation is not required. However if the distance is required to exceed a threshold value, then normalisation is essential.

It should be noted that Cunningham's term "direct metric" is not necessarily synonymous with the notion of *metric* in the mathematical sense. In order to be a metric, a measure of similarity *d* between objects *a* and *b* must conform to the following four rules:

- non-negativity: $d\left(a,b\right) \geq 0$
- identity: $d\left(a,b\right) = 0 \leftrightarrow a = b$
- symmetry: $d\left(a,b\right) = d\left(b,a\right)$
- triangle inequality: $d\left(a,c\right) \leq d\left(a,b\right) + d\left(b,c\right)$

The measures of similarity described above can be used in either metric or non-metric problem spaces. In metric spaces, some optimisations are possible; the number of comparisons between objects is reduced because the triangle inequality is followed. For example, the *fish and shrink* [62] algorithm involves "fishing", i.e. random access to the database. The "shrinking" refers to a narrowing interval (range) of possible distances between the query case and cases that are not yet tested. As a result of the shrinking, cases that are very different from the query can be excluded from consideration by the retrieval algorithm, and this has the potential to dramatically speed up retrieval.

Many works in the literature assume that the problem space is a metric space. However, Tversky's seminal paper [63] questioned the assumptions which are necessary for metrics (see above). In particular, psychological experiments have shown that human scores of similarity are often not symmetric. Sometimes, working in a metric space is an approximation. For example, route planning is not a metric space when there are one-way streets or motorway junctions that permit only either exit or entry. Despite this, most tables of distances between cities assume symmetry, since it is an acceptable approximation. So, the optimisations that metric spaces bring must be balanced against the need for the distance measures to accurately reflect reality.

The biggest issue with the direct metrics, however, is that the features or attributes of a case usually vary in their importance or relevance to the retrieval process. So, any method which treats the attributes

Likert	*Description*	*Weight*
1	“Irrelevant”	0
2	“Not important”	0.25
3	“Sometimes important”	0.5
4	“Important”	0.75
5	“Very important”	1

Table 2.4: An example of the use of Likert scales for weights

as equal may provide misleading results by over-emphasising differences in attributes which are of minor importance, or under-emphasising differences in attributes which are of major importance. This problem can be resolved by using a *weighted sum* approach, in which *weights* are assigned to the features; the magnitude of the weight reflects the importance of the feature. For example, the Manhattan distance can be reformulated using w_r, the weight of the *r*th feature, as follows:

$$distance\left(x_1.x_2\right) = \textstyle\sum_{r=1}^{n} w_r|f_r\left(x_1\right) - f_r\left(x_2\right)|$$

Symbolic attributes present a problem, since the measures described above can only deal with numeric values. Mono-valued attributes can be dealt with using a matrix which contains numeric values for the distances between symbols. If it is a true metric, then for a group of n symbols, $\frac{n}{2}\left(n-1\right)$ weights must be provided: presumably by a domain expert. However, this does not help for multi-valued attributes. In this situation a metric based on set theory may be applicable, such as some formula involving the cardinality of the intersection of the values in the two cases [2]. If the representation consists of a mix of numeric, mono-valued symbolic and multivalued symbolic attributes, then a mix of local similarity measures may be needed, which can then be incorporated into one global measure.

If weights are required, the obvious way of gathering them is to ask a domain expert to provide the values. However, the concept of numerical weights is not intuitive and so this is fraught with difficulty. Also, different domain experts may provide different weights. One alternative to this is to use a Likert [64] scale, or linguistic terms which map to the weights; I have given an example this in table 2.4.

Instead of consulting human experts, several methods of automatically learning weights are in existence. Wettschereck and Aha [65] made the point, as do many other authors, that k-NN methods are sensitive to feature sets “containing irrelevant, redundant, interacting or noisy features”, and suggested the use of weights as a solution to these problems. They described methods for learning the weights which utilised no domain-specific knowledge. Feature weighting methods were distinguished along several dimensions, including *model*. The model dimension was split into wrapper and filter methods:

- *Wrapper* methods utilise performance feedback. Weights are modified by incremental hill climbing, genetic algorithms, or the “variable kernel similarity metric” which utilises knowledge of the gradients of functions.

- *Filter* methods do not utilise performance feedback. One type of filter method sets weights using conditional probabilities to assign large weights to features which are correlated to a given class. Alternatively, the "value-difference metric" computes similarity between individual feature values, then assigns larger weights to features which have a skewed distribution across the classes. A third method uses "mutual information", which is defined as the "reduction in uncertainty of one variable's value given knowledge of the other's value".

Wettschereck and Aha tested these methods on standard datasets and found most methods worked well unless there were "highly interacting features". They stated that wrapper methods learn faster and are more accurate than filter methods. An authoritative paper [66] states that the methods described in Wettschereck and Aha are not applicable when there are complex representations. This is not surprising since a complex representation will typically involve interacting features, and a higher number of features.

Cunningham [59] describes *transformation based measures* as those that assess the similarity between objects as being the effort required to transform one object into another. One example is the *earth mover distance* [67] for image similarity; it uses linear programming and works using an analogy of transforming mass from one distribution to another. Transformation based measures include some domain-specific specialist measures such as the *Levenshtein Distance* [68] for text, which counts the number of insertions, deletions and substitutions required to transform one string into another.

Similarity measures for graphs tend to utilise *edit distance*, the *largest common sub-graph*, or *sub-graph isomorphism*. Unfortunately many of these problems are NP-complete, although heuristics are available to make them more tractable. In [69], several similarity measures were used, e.g. a count vertices that are in common, or the maximum matching common subgraph. In [70], in order to solve the graph isomorphism problem, a library of commonly occurring subgraphs (referred to as "models") was maintained. The problem was then reduced to trying to find an isomorphism between an input graph and a model graph; this was done using decision trees. The worst-case run-time complexity of the algorithm was quadratic with respect to the size of the graph. However, the size of the decision tree grew exponentially, and in my opinion this is a serious disadvantage of this approach since it is likely to be unacceptably inefficient for complex cases (which involve larger graphs).

The most common of Cunningham's *information theoretic measures* [59] is *compression-based similarity*. Using this measure, the compressed size of a document *a* which was compressed using the codebook from document *b* gives an indication of the difference between documents *a* and *b*. The phrase codebook refers to the dictionary that is used in methods such as LZW [71]. Compression-based similarity is simple to understand and easy to apply. It requires neither training nor domain knowledge and has been shown to be effective for many applications, e.g. the detection of software plagiarism [72] and spam filtering [73].

The final category referred to by Cunningham [59] is *emergent measures*. This is a loose collection of

	no training	*training required*
knowledge light	(1) • Levenshtein Distance • Earth Mover Distance • compression based	(2) • weighted sum with learned weights • graph edit distance • random forests • other algorithms involving collections of decision trees
knowledge intensive	(3) • weighted sum with automatically learned weights	(4) • hybrid approaches, e.g. in Houeland [75]

Table 2.5: Classification of similarity measures

more recent algorithms which are designed to exploit the computing power of modern computers. One such algorithm is *random forests* [74]. A random forest consists of a collection of decision trees which have been constructed by a stochastic[12] process. The trees are not pruned (as would normally be the case), since using unpruned trees increases diversity. The output consists of the modal value[13] from the decision trees.

Random forests are a generic type of classifier; several researchers compare random forests favourably with other classifiers such as neural networks. An interesting example of the use of random decision trees in CBR is given by Houeland [75], where the techniques were applied to the domain of palliative care. Houeland used binary trees of depth 5; each node involved a the range of a numeric attribute being split in two; the leaf nodes consisted of a set of cases from the case base. The 'knowledge-light' approach involving decision trees was compared to a more 'knowledge-heavy' approach where domain experts were asked to consider which features of the case were relevant. A hybrid approach which involved elements of both of these was also tested. The results showed that the 'knowledge-light' approach outperformed the 'knowledge-heavy' approach in this domain.

A wide variety of similarity measures has so far been discussed, but these are only examples and many other methods exist in the literature. So, for a CBR system implementer the important question arises: which similarity measure is the best for a given case base? To facilitate this discussion, I have divided similarity measures into four categories in table 2.5.

[12] In this thesis, "stochastic" refers to any process which is non-deterministic, such as the output of a pseudo-random number generator.

[13] The mode is defined as the most common value.

The most important distinction is between *knowledge light* (meaning little domain knowledge) and *knowledge intensive* (extensive domain knowledge) measures. Knowledge light approaches offer a distinct advantage over knowledge intensive approaches, but are not always applicable. Knowledge intensive approaches can be fraught since, in general, the acquisition of domain knowledge is difficult; knowledge is difficult to obtain and, may not be accurate or relevant. Acquisition difficulties arise due to the "knowledge elicitation bottleneck", which is discussed in more detail in section 2.4.3. Experts may be skilled in the construction, deconstruction, and repair of a particular artefact, but this does not necessarily mean that they can give a reliable numerical score as to how important a feature is for similarity in CBR. If weights are forthcoming, they tend to be very subjective. For example, in the construction of an automated Scrabble system, Sheppard [76] found experts overestimated the importance of some factors, leading him to "treat expert guidance with scepticism". Knowledge intensive approaches should only be used if knowledge-light ones have been shown to be inappropriate for the problem in hand, or if there is good access to high quality domain knowledge (an atypical situation).

Some knowledge-light approaches do not require training: see (1) in table 2.5 e.g. Levenshtein Distance for text. If the data supports such an approach, and it performs well for the task at hand, then the lack of training and lack of domain knowledge would make these sorts of measures an obvious choice.

If none of the measures in category (1) are suitable then the decision will be guided by whether or not suitable training data is available. The measures in category (2) all require training data: and, the more complex the case is, the more data will be required. Complex cases tend to have more attributes, and since the problem space has a large number of dimensions, more and more cases will be required to adequately train the system. Assuming adequate training data is available, then the methods in category (2) may be the best choice.

If categories (1) and (2) are excluded, but a domain expert is available, then the obvious alternative is category (3), that of asking the domain expert to provide weights for the features. If the case representation is feature-vector then this is straightforward to implement. Otherwise, features will have to be inferred or calculated from the case representation. For example, object-orientation has certain generic features that could be extracted, such as the depth of the inheritance tree. However, domain-specific measures could be valid as well, such as the number of "child" objects in a particular "has-A" relationship. Of course, as they are typically implemented, objects can have numeric values, and a domain expert may be able to distinguish which of these are important enough to be considered as "features" for similarity purposes.

Category (4) is included for completeness since a hybrid approach showed a slight improvement in results in [75], when compared to results from categories (2) or (3) alone.

If none of the options in table 2.5 are appropriate, then the only applicable strategy is to build up the case base by non-CBR means (e.g. via model-based reasoning or rule-based reasoning) until it reaches a size when training is possible for the measures in (2). This pathway may also be viable if category (3) was used at the beginning, since it may be possible to use the expert's weights as a starting point for improvements.

Effectiveness is one criterion for choosing a similarity measure. The measure must be effective otherwise adaptation may be too difficult or impossible. *Efficiency* may also be a factor; for example it is said in [59] that feature-vector approaches tend to be considerably faster than compression-based ones. However, efficiency is not always a significant factor. Techniques such as fish and shrink [62] can be used to avoid having to search the whole case base. Also, modern computers are sufficiently powerful so that the option of a simple linear search of the case base may be possible in an acceptable time.

Various methods of similarity have been discussed; some are specialist whilst others are quite general, some are simple whereas others are complex, and some require domain knowledge or training and others do not. The choice of an appropriate and effective similarity measure is important, but just as important is how that measure is used to retrieve cases; this will be discussed in more detail in the next section.

2.2.2.3 Putting it all together: implementing retrieval

Once an effective similarity measure has been chosen, the implementer of a CBR system must then decide how to use it in the retrieval process. A key decision is whether or not it will be necessary for the retrieval algorithm to search every case, or if it can be restricted to a portion of the whole case base. Also, it is possible to employ more than one criterion in the search, as just a single similarity measure may not be optimal. Another issue is that it may be desirable to retrieve more than one case, and this may prove useful in coping with adaptation failure.

If a case base is not large, then with the sort of impressive computing power that is cheaply available nowadays, it may be feasible to do a straightforward linear search of an entire case base in an acceptable time. However, if this is not possible, then one option is to use indices, as described in section 2.2.2.1. For example, a balanced binary tree could select around 100 cases from a case base of 100000, by traversing just 10 nodes of the tree.

Cunningham [59] lists several other strategies for making retrieval more efficient, e.g. *Fish and Shrink* [62], which is discussed in the previous section. *Cover Trees* [77] are a way of organising the case base to avoid having to search through every case. Both Fish and Shrink and Cover Trees require the similarity measure to be a true metric. *Footprint-based retrieval* [78] uses decision trees in a two-stage process; the first stage is to localise the search, and the second is to find the nearest case to the query case within that localised section of the case base. Footprint-based retrieval is not guaranteed to return optimal results, but remains an effective heuristic.

Cunningham [59] discusses a technique known as *case retrieval networks* (CRN), which can be configured to return the same cases as k-NN. In a CRN a case is said [79] to constitute a series of information entities (IEs). An information entity could be, for example, an attribute-value pair; CRNs depend on the case representation being feature-based. The network has two types of nodes: case nodes and IE nodes. The nodes are connected by two types of arcs: relevance arcs, and similarity arcs. A relevance arc connects a case to the IEs which constitute its representation, whereas a similarity arc connects two IEs. For example the IEs "Price: £10" and "Price: £10.50" might be connected by a similarity arc since the attribute is the same and the values are similar. Case retrieval is effected by "activating" the IEs in the query case, and propagating this activation through the network. The networks can be enhanced through the addition of domain knowledge. It is claimed [79] that maintenance of the case base is straightforward, so CRNs appear to be an attractive approach for speeding up retrieval.

One key factor which determines the *efficiency* of retrieval is the computational efficiency of the similarity measure. However, for *effectiveness* it is sometimes desirable to use more than one similarity measure. For example, in [80] there were two phases: retrieval and *validation*. The latter involved applying a series of tests, and in this work, all the tests needed to be passed. However the authors pointed out that weights could be applied to the test results and then the success criterion would involve the sum of the weights of the successful tests exceeding a preset threshold value. The tests were supplied with the assistance of domain experts. When validation was tested in real-world domains, it was found that the tests were often not self-contained; they were inter-related in some way, so domain experts were used to group the tests into sets and a dependency graph was constructed. It was claimed that this approach is relevant to design tasks: retrieval would work on the *surface features*, which means the design specification. The validation would consist of checking that the new design meets the specification. The most interesting feature of this work was the use of different levels of detail: surface features are used first, which is presumably very efficient. Secondly, there is an examination of the more complex 'deeper' features of the case, which can make retrieval more effective, since domain knowledge can be employed.

Börner [81] proposes a process which makes an assumption that cases consist of two parts: the *surface* or (attribute-value based) part, and the *structural* part (this assumption is reminiscent of the work in the preceding paragraph). The surface representation can be transformed into the structural via an invertible transformation function. This means that each surface representation corresponds to one structural representation, and vice versa. The process begins with a "surface similarity assessment", which retrieves a set of cases which are similar to the query case by comparing the surface parts. This is efficient due to the simplicity of the surface features. As well as a case base, a rule-base exists and each modification rule within it has an inverse. Through careful selection of rules and application of their inverses, both parts of the solution case are obtained. The surface representation can be shown to the user for their approval; the user validates this and the whole process ensures that only cases which can usefully be adapted are retrieved.

Börner makes the distinction between analysis and synthesis tasks. The claim is that for synthesis tasks, similarity is not enough: the adaptability must be determined. One possible explanation for Börner's claim is that, if the cases are complex artefacts, that the dimensionality increases and the mapping between problems and their solutions becomes more complex. The assumption that "similar problems require similar solutions" breaks down slightly. There are obvious flaws in Börner's proposal: rules are not always invertible, and acquiring them can be an onerous task. However, for complex synthesis tasks, the arguments about adaptability are persuasive. Also, the idea of representations on multiple levels with different similarity measures is an interesting one, since it can achieve an effective trade-off between thoroughness and efficiency.

Much of the approaches in this section have involved using similarity measures to guide the retrieval algorithm. However, concerns about adaptability have lead some researchers to consider an alternative to solution-guided retrieval: *adaptation guided retrieval* (see section 2.2.3.2). Smythe and Keane [82] argue that retrieval based on similarity of "surface" attributes is often insufficient, and that it needs to be augmented by "deep" domain knowledge about adaptability to ensure that retrieval is accurate. Their algorithm is able to recommend adaptation methods for a particular case; the methods carry a score which correlates with their complexity. The adaptation cost of the case is the sum of the cost of each of those adaptation methods. The notion of "adaptation cost" is used in retrieval to prioritise cases which are easy to adapt - and eliminate those which are impossible to adapt. An important advantage of this approach is that it minimises the likelihood of adaptation failure.

The approaches of both Börner and Smythe and Keane involve a merging of retrieval and adaptation. If such notions are cast aside and the activities are viewed as separate as per the traditional 4 REs model (see section 2.1.1), then one factor that must be considered is the goal of the retrieval algorithm:

1 If the goal is to *retrieve just one case*, then this is presumably the one with the highest similarity to the query case (or alternatively the first one with a satisfactory similarity).

2 The goal may be to *retrieve a small cache of cases*, but with an end goal of *reusing and adapting one case*.

3 Lastly, the goal may be to *retrieve and reuse several cases*.

Option 3 implies that the CBR system will perform *case merging*. In many domains with complex cases, the issues of dimensionality in the problem space are such that it is unlikely that there will ever be sufficient coverage of the case case for one single case to be retrieved that is a satisfactory match. However, sometimes it is possible to retrieve two or more cases, and reuse only parts of those cases; the parts are merged together to form one coherent case that is the solution to the problem at hand. The parts are (presumably) simpler structures than the whole, and this addresses the issues of dimensionality: it is much more likely that a partial match will be found than a complete match. Case merging is discussed in more detail in section 2.2.3.3.

The choice between options 1 and 2 determines the CBR system's strategy for dealing with failure. Two sorts of failure are applicable: in null adaptation systems, it is possible that the retrieved result will be unsatisfactory to the user, and they will then presumably expect to be offered an alternative. This is particularly true in a *recommender system*.

In systems with adaptation, the possibility of adaptation failure is potentially the weakest point. If option 2 was chosen, then the CBR system has the option of choosing another case from the cache. If option 1 was chosen, or the cache from option 2 is exhausted, then the CBR system has several options for dealing with the situation:

- *Run the retrieval algorithm again*. Obviously, the case(s) originally retrieved would have to be excluded. If there is a threshold similarity value for retrieval, this might have to be decreased.

- *Switch to case-merging*. Even systems which are described as using case merging are likely to try and adapt a single case first, since this is the easier option. Thus, case-merging is normally a backup strategy.

- *Switch to another problem-solving methodology*, e.g. rule-based. Hybrid systems are featured in section 2.3.

- Finally, the system may have to resort to reporting to the user that CBR has *failed*. Many of the early CBR systems (see section 2.1.2.2) were able to learn from failure.

This discussion has shown that there are a wide variety of methods to measure similarity and to effect retrieval in CBR. The methods vary in their efficiency and their effectiveness. Efficiency considerations are only valid if response times are unacceptably slow. Effectiveness, however, is a key issue and since the methods are typically not universally applicable (e.g. some only work for feature vector representations), they must be chosen with the task and the case representation borne in mind. In many CBR systems (in particular design), retrieval alone is insufficient. Presenting a user with a previously solved problem is not always useful, since the user's goal is solving their current problem. In those systems, the heart of CBR is *adaptation*.

2.2.3 Adaptation

In the adaptation stage of CBR, a copy[14] of the retrieved solution is altered so that it meets the requirements posed in the new problem (query case). When CBR is working well, a solution which is *very similar* (but probably not identical) to the query case is retrieved. The assumption is that, because the problems are very similar, the solutions will also be very similar (see section 2.4.1). Adaptation 'tweaks' the solution from the retrieved case to make it suitable for the query case.

[14] The original solution is typically retained without modification.

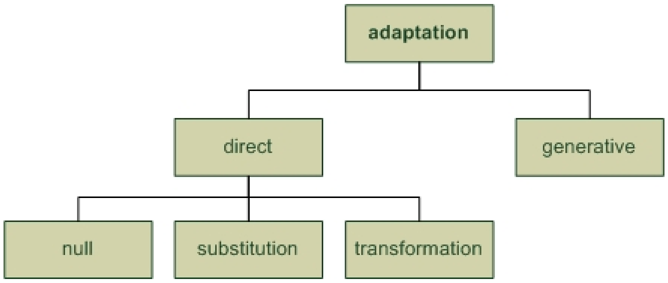


Figure 2.6: Taxonomy of adaptation operators

In [66], two types of adaptation are distinguished. *Adaptation during reuse* happens when adaptation is employed during the construction of a solution. In contrast, *adaptation during revise* is described as occurring after solution is proposed, when feedback indicates that it is unsatisfactory and needs to be repaired. In my opinion, this distinction is unnecessary and in this thesis, adaptation is considered synonymous with the revise step of the 4REs model [1].

The ideal situation is where the case-base contains an existing solution that can be reused without modification, meaning adaptation is not required. However, in many CBR systems this situation is a rarity: the coverage of the case base is too sparse to make it likely that *exactly* the same problem will be encountered twice. Adaptation can thus be seen as a way of making up for lack of coverage of the case base. Figure 2.5 provides a good visual illustration of this: there is a negative correlation between the density of the case base and the mean "depth" of adaptation. Adaptation is often a particularly difficult part of CBR; if the distance between retrieved and query case is large then extensive changes may be required which can be complex or even impossible to accomplish.

This section will begin with an introduction to the three adaptation operators. These operators can be combined and used in a myriad of different ways, and some of these strategies are presented, alongside relevant examples of their use. Then case merging, and adaptation in general are discussed.

2.2.3.1 Adaptation Operators

All adaptation is accomplished through the use of three basic operators: substitution, transformation and generative adaptation. Since there is no universally agreed terminology[15], I have devised a taxonomy of adaptation operators which is shown in figure 2.6.

In *direct adaptation*, the existing solution from the case base is used directly to meet the new requirements. In contrast, *generative adaptation* reuses the method which was employed to create the existing solution. This method is applied to the new problem. Generative adaptation is usually considered to be synonymous with derivational replay, which was discussed briefly in section 2.2.1.2. However, a new type

[15] For example, Bergmann and Wilke [12] refer to what I call direct adaptation as "transformational adaptation" and transformation is described as "structural adaptation". This is inconsistent with Kolodner's [57] definition of transformation, which is distinct from substitution.

of generative adaptation is introduced in chapter 7.

Null adaptation is the simplest option, where in fact there is no adaptation! Sometimes this is because the task does not require adaptation. For example, in spam filtering [83] the solution space normally has just two possible values (e.g. "spam" or "not spam"). Adaptation implies a slight "tweaking" of the retrieved case, so it is clearly irrelevant to this sort of task. Alternatively, adaptation may be omitted because it is left to the user to perform manually; a noteworthy example is Clavier (see section 2.2.3.4).

Substitution is where a part of the solution is replaced with another object or value [57]. For example, in a meal planning system, if the retrieved case was 'beef curry' but the new case was a vegetarian meal, then then tofu could be substituted for the beef. Kolodner [57] defined several sub-categories of substitution, for example *local search* uses a replacement for an (unsuitable) item is found by searching an abstraction hierarchy. In *reinstantiation* the roles in the new solution are filled differently to those in the solution part of the retrieved case. For example, a dispute involving ownership of an orange was solved with reference to a previous dispute with a candy bar. The orange in the new case is performing the same role as the candy bar did in the old case.

Parameter adjustment is where interpolation is used to change a value in the retrieved case so that it better matches the new solution, for example in criminal justice, the sentence for a crime might be altered if its circumstances of the new crime (query case) differ from the previous crime (retrieved case). Parameter adjustment may be particularly suited to numerical attributes, whereas local search could be used for symbolic attributes.

Transformation involves structural changes to the solution, such as the insertion and/or deletion of parts. In the meal planning example, this could be to delete an undesirable ingredient, for example the removal of Brussels sprouts from a traditional English roast dinner if they are not required. Transformation may be guided by a model - see below for a discussion of model based adaptation.

It is clear that transformation is a more powerful and flexible operator than substitution. Often, the two are combined, and adaptation may consist of several substitution steps, several transformation steps, or a mixture of both. Sometimes this is because more than one change is required. Another meal planning example is given in Kolodner [57] where lasagne is adapted so that it becomes kosher. In this situation, two substitutions were required: tofu-cheese for cheese and non-dairy margarine for butter.

In some cases it may be that only one (or few) changes are required, but when implemented these solutions render the solution invalid, necessitating further changes. A simple example in judicial sentencing may occur if the crime in question (the query case) has more aggravating circumstances than the previous crime (retrieved case), which was punished by a term of imprisonment. Therefore, the solution is obtained by parameter adjustment: the term of imprisonment is increased, using some heuristic. However, this then poses a problem: the maximum term of imprisonment has been exceeded, so instead the sentence is reduced to the maximum term and an additional fine is imposed. Thus, an initial substitution necessitated an additional substitution and a transformation step: the addition of a fine to compensate for the reduction in the prison term.

Many other awkward situations can be envisaged, for example in the meal planning domain it is possible that a substitution will result in two incompatible ingredients being present. These examples indicate that adaptation can be a complex process which is not always guaranteed to succeed.

2.2.3.2 Strategies for adaptation

This section discusses some ways in which the adaptation operators can be used and combined. Where possible, examples are given as to where these strategies have been employed; these examples are largely drawn from [66].

Model based adaptation is often implemented when a causal model is available but is incomplete, which is why it is augmented with CBR. CASEY [20, 21] was an early example of this; it built on the work of the Heart Failure Program, which reasoned according to a causal model. A solution had three parts: diagnosis, therapy and explanation. CASEY had strategies to "repair" each of these if the solution in the retrieved case is unsuitable. For example, the therapies could be repaired using the knowledge that certain therapies benefit certain physiological states.

Less commonly, the primary motivation for model based adaptation is the intractability of the model. In FormTool ([29], see section 2.1.2.3), the solution in the retrieved case is used to provide the starting point of a search. The solution space is so large that without this starting point, an exhaustive search would be impractical. The examples of CASEY and FormTool show that even if a model is flawed, it still may be of use when incorporated into a case based reasoner.

Evolutionary algorithms have been employed for adaptation. For example, GENCAD [84] used a genetic algorithm to adapt floor plan layouts according to the principles of feng shui, the "Chinese art of placement". Cases in GENCAD were the designs of such layouts. Adaptation was performed by two operators which made changes to the designs: crossover and mutation. *Crossover* was used to combine aspects of two designs; the resulting "offspring" were two new designs. *Mutation* altered one design to produce a single new design as its offspring. The algorithm iterated through a cycle: a population of designs was subjected to crossover and mutation. The designs were evaluated using a *fitness function*, which was formulated using the principles of feng shui. The best designs of the generation were chosen and the cycle repeated until a satisfactory design was produced. The stochastic nature of the process produced some surprising results, which the implementers could not have anticipated. This is surely a key advantage of evolutionary algorithms: their results may be quite creative or innovative.

One of the drawbacks of using evolutionary algorithms is the need for a fitness function. All problem-solving techniques require some measure of success, whether this is done automatically or manually via expert users. However, in the case of genetic algorithms the fitness function will typically be applied many times, and so asking users to evaluate the solutions manually is typically not feasible. The efficacy of the algorithms is often determined by the accuracy of the fitness function.

Adaptation guided retrieval was introduced in section 2.2.2.3. The ease of adapting a solution is used in the retrieval algorithm, with a goal of reducing or eliminating the possibility of adaptation failure. It is a technique which has been applied to Déjà Vu, which automated the design of software that controlled robotic devices in steel mills [82]. Déjà Vu produced "solution charts" for the programs, and adaptation therefore involved modification of those charts. The knowledge needed to perform such modification was stored in so called "specialists" and "strategies". Strategies were more general than the specialists. Specialists performed the basic adaptation mechanisms of insertion, deletion and substitution within the graphs, without any knowledge of the effects of other specialists. Strategies coordinated the specialists; their operations had interdependencies and therefore the order in which the specialists were invoked was significant. They also managed conflicts that arose between the specialists: for example, invoking an earlier one may cause the preconditions of a latter one to not be met.

Both specialists and strategies had "action knowledge" encoded: this referred to the actual modifications that are performed. In fact all knowledge intensive adaptation methods need such "action knowledge". However, in Déjà Vu they also had their "capability knowledge" encoded; this defined their scope and was used to guide retrieval so that only adaptable cases were retrieved. This work is significant both as a proof-of-concept for adaptation guided retrieval, and because of the way in which complex multi-step adaptation is controlled.

Adaptation is presented as a *constraint satisfaction problem* in [85]. It uses a heuristic called the *minimum conflicts repair algorithm* to resolve constraints in design specifications. The values in the retrieved case are used as the starting point for the algorithm. Then, using the constraints from the query case, an iterative repair process progressively resolves any conflicts; when there are no more conflicts, the solution is found. It is worth noting that not all problems involve constraints that can be explicitly encoded. For example, design problems (see chapter 3) often involve constraints that are difficult to encode in a computer system, because they may be vague, conflicting and rapidly changing.

As in many other aspects of CBR, approaches to adaptation differ in the amount of domain knowledge that is relied on. In knowledge heavy approaches, the adaptation knowledge is explicitly encoded in the system, e.g. model based adaptation. Knowledge light methods involve some form of automated learning, such as evolutionary algorithms. Craw et al [86] explored the use of decision tree algorithms to acquire adaptation knowledge. "Adaptation training examples" were selected from the case-base using a *leave one out* strategy. One algorithm used by Craw et al was the C4.5 algorithm, which chooses nodes which allow the highest information gain. C4.5 works in a "general-to-specific direction", i.e. as the nodes in the tree are traversed the adaptation knowledge becomes more specific. Many of the concepts in the discussion about retrieval (in section 2.2.2.3) apply here; for example decision trees, as per all knowledge light methods, require sufficient data to train them.

Another knowledge light method was proposed by McSherry [87]. What makes this approach remarkable is the claim that knowledge-light adaptation can be performed using just 3 cases. One of these, chosen because of its similarity to the target, plays the role of the retrieved case in the classical 4REs model [1]. The other two provide the adaptation knowledge using simple heuristics: the values of attributes in these two cases are compared and the differences between them are applied to the retrieved case to give the solution. Although this technique sounds promising, it requires the retrieved case to differ from the target case only in the value of one attribute. In many problem domains, this would be wildly unrealistic.

Case based adaptation is an approach that is similar to McSherry's work, was was used in DIAL [88]. Two cases were retrieved from the case base: one plays the role of the retrieved case in the 4REs model [1], whereas the other is the *adaptation case*. The adaptation case will be similar to the query case, and will contain a previous example of successful adaptation. The process which was applied in the adaptation case is re-applied to the retrieved case.

The wide choice of approaches listed here indicates that the choice of adaptation strategy has to be a pragmatic one, tailored to the needs of a particular problem. If a model exists and can be encoded into a computer system, then model based adaptation may be a good choice. A CBR system that incorporates model based adaptation is a special case of a hybrid system (see section 2.3). Alternatively, if the adaptation is primarily about solving well-defined mathematical problems such as constraint satisfaction then many algorithms exist for this and little or no domain knowledge may be required.

If creativity (in the output) is an advantage, and a good fitness function is available, then evolutionary algorithms may be considered, since they sometimes produce surprising solutions. However, if the problem is more about interpolation that creativity, and the problem case coverage is very good, then knowledge-light methods should be explored. Adaptation functions can sometimes be learned through analysis of the case-base, but it must be borne in mind that it tends to be very difficult to get dense coverage of a case base when the problems exhibit high dimensionality.

Some systems utilise more than one strategy for adaptation, e.g. DIAL [88], which was capable of using both cases and rules for adaptation, as well as allowing manual adaptation. Of course, adaptation does not have to adhere to any of the strategies listed here. For example, Kolodner [57] refers to *commonsense transformation*: adaptation commonly uses heuristics and other domain knowledge. However, the disadvantages of systems which incorporate domain knowledge are well known: acquiring such knowledge can be difficult, and the resulting heuristics are not guaranteed to be optimal. In some cases the best strategy is to abandon the idea of retrieving a single case, in favour of case merging.

2.2.3.3 Case merging

Case merging is a variant on the traditional 4REs model [1] where multiple cases are retrieved instead of one case. Parts of the retrieved cases are then combined in some way to provide the solution. It is also known as *compositional adaptation* [12]. As discussed in section 2.2.2.3, case merging is a method of coping with poor coverage in the case base. Problems of coverage are typically more acute when the cases are more complex, due to the high dimensionality.

> "The power of CBR is severely curtailed if problem solving is limited to the retrieval and adaptation of a single case. For complex problem domains, it is unlikely that a single case will be available which closely matches all of the target problem details, and hence sophisticated adaptation support will be necessary. However, the same problem may be more readily solved by combining parts of many different solutions, without the need for the same level of sophisticated adaptation. For this reason, the strategy of reusing multiple cases is immediately appealing." - Smyth et al [44].

CELIA [45] was an early CBR system which performed automobile troubleshooting, using case merging. CELIA stored cases in pieces called "snippets", and the integrity of the solution was preserved by links between the snippets. Snippets could be retrieved in two ways: by following the links from one snippet to the next, or directly, via a weighted similarity metric. The former was given a higher precedence than the latter since the authors stated they wanted to retain the coherence (of the cases).

The examples given in [45] show CELIA retrieving a snippet, and following the links to other related snippets; it is only when a retrieved snippet is found to be unsuitable that the global search is carried out. A snippet is deemed unsuitable if its predictions (about the automobile) are inconsistent with reality. The author pointed out that, even if a snippet was small, it was not merely a rule, since the snippets carried links to one another and also stored the context in which they were applied. So CELIA's case representation was *not* simply about inferring rules from cases. It was mentioned that storing the context could facilitate reasoning that would prohibit the retrieval of incompatible snippets. In contrast, in a rule-based system, the rules are not traditionally linked to one another and do not normally store contextual information, therefore if rules are retrieved there is no way to ensure their compatibility. The example of CELIA raises some interesting questions and choices about how to implement case-merging.

When to use case merging versus a single case?

CELIA [45] prioritised retrieval of a single case (via its chain of related snippets) over retrieval of multiple cases. In my opinion this is the correct decision, as the simplest solutions to any problem should always be tried before the complex ones; case merging has disadvantages over single case retrieval due to its inherent complexity. However, the right balance must be struck: retrieving two cases which are a good match may be better than one case which is a poor match.

How are cases stored?

As discussed above, CELIA [45], stored cases in parts called "snippets". The key advantage of this is that the parts can contain links to each other which can be used to intelligently aid retrieval. Déjà Vu [44] also stored cases in parts, in a hierarchical structure. Such storage may result in a reduced storage requirement if the parts are reused, but this is rarely significant with modern computer systems.

CADET [46, 89] constructed designs for hydro-mechanical systems. Cases in CADET were stored as a whole, and not fragmented. The justification for this was that the components of a design can have multiple functions, and the goals for a complete system can change during the design process, so indexing components by their subgoals could be problematic.

If case merging takes place in a CBR system which stores cases as a whole, then the question arises of how to break the case up and access the parts. If a case is stored in parts, then it is important that the integrity of a case is maintained (e.g. by storing links between parts of a case).

> "In the case of all things which have several parts and in which the totality is not, as it were, a mere heap, *but the whole is something beside the parts*" - Aristotle (italics mine).

What size are the parts that are retrieved?

If the cases are stored as a whole, then perhaps the better question is about demarcation: *how can we identify parts of a case that can be reused?* There is no general answer to this, what is an appropriate granularity for one problem may not be so for another. Ideally the parts should be as self-contained as possible to minimise interactions.

COMPOSER [85] was a case based design system which represented cases as constraint satisfaction problems. Multiple cases were retrieved; the values from these retrieved cases were used to initialise the values of the corresponding variables in the new problem. The problem decomposition algorithm utilised knowledge of the case base; thus the size of the sub-problems was determined by the composition of the case base.

CADSYN [90] used "generalised decomposition knowledge" to break down designs. This decomposition knowledge originated in a 1980s frame-based system [91] which represented designs using domain knowledge in the form of preconditions, constraints and goals. In addition, alternatives to those goals were stored, as well as an ordering for the goals. When a goal could not be directly satisfied, the problem was decomposed. Thus, the degree of decomposition was determined by the nature of the problem, which seems a pragmatic approach.

Are the cases of equal importance?

In CELIA [45] several cases can be retrieved, and although their contribution to the final solution may not be equal, there is no concept of any one case in particular being more important than the others. Kolodner [57] speaks about *case based substitution*, as a method of adaptation[16]. In case based substitution, a value in the retrieved case is substituted for one in another case. Thus, it is possible to construct a system where the solution comes principally from one case, but additional cases are used to perform minor modifications to it.

In Déjà Vu [44], there are two sorts of cases: abstract cases (which offer "high-level" general solutions) and concrete cases (which contain real, specific solutions). A graphical representation based on "structure function charts" is used for both types of cases. An abstract case typically corresponds to several concrete cases. Since matching (for retrieval) is done against the abstract cases, they are used as a form of indexing. The abstract cases store decomposition knowledge; both concrete and abstract cases may be retrieved but decomposition only occurs if an abstract case is reused. Their role in indexing and decomposition arguably makes abstract cases more important than concrete cases.

How can the retrieval of incompatible parts be resolved?

This point is particularly important since the main disadvantage of case merging is the possibility if incompatibility between the parts. Case merging is particularly appropriate to complex tasks such as planning and design, since these are the sort of tasks that the issues of high dimensionality tend to affect the most. In design, it is possible to retrieve parts which are spatially incompatible with one another. It is also possible to introduce redundancy in the case merging process. In planning, additional to these issues there may be temporal problems: an earlier step may cause the pre-conditions of a latter step to not be met.

There are two obvious strategies for dealing with incompatibility: prevent it from happening, or repair it once it has happened. Intuitively prevention seems better than repair, however if the component parts have complex interactions with each other, then it is possible that prevention will not be feasible as it may be difficult to foresee those interactions. For CELIA [45], the danger of incompatibilities was discussed but no strategy was implemented. In COMPOSER [85], the minimum conflicts heuristic is responsible for management of incompatibility, and for adaptation in general. Graph-theoretic heuristics which employ tournament selection have also been successfully employed to help merge cases in CBR systems for course timetabling problems [92]. The tournament selection prioritises courses which have the most constraints to be scheduled first.

[16] Note that case based substitution is not the same as *case based adaptation*, which is discussed in section 2.2.3.2. The use of cases to acquire adaptation knowledge is not (in my opinion) case merging, even though more than one case is retrieved. This is because the additional cases are not structurally a part of the solution, they are merely used to guide the adaptation process. However, this distinction becomes blurred when there is case merging in derivational replay.

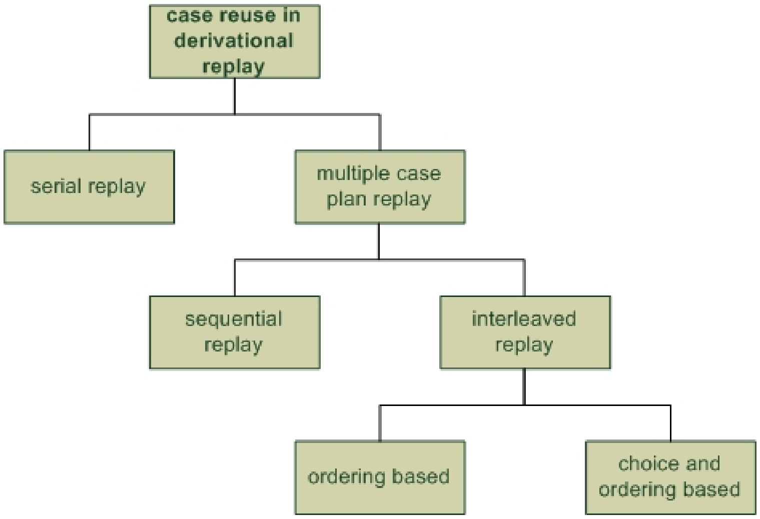

case reuse in derivational replay
serial replay
multiple case plan replay
sequential replay
interleaved replay
ordering based
choice and ordering based

Figure 2.7: Taxonomy of strategies for multiple case reuse in derivational replay

As well as incompatibilities, it is possible that when the retrieved cases are combined there will be gaps or missing parts in the solution. Many CBR systems for planning and design often incorporate some sort of non-CBR reasoning (e.g. rule based or model based). This reasoning is often used in adaptation, and if present it could be of use to repair inconsistencies between merged parts, and to provide missing parts to solutions.

The examples above have shown how case merging has been applied in design; however it is also used in several case based planners. Muñoz-Avila and Cox [36] make the point that case merging is most likely to be successful if the planner's task involves solving a set of goals and the case base consists of cases that can be decomposed into parts that can be combined in different ways to solve different goals. Strategies for case merging within derivational replay (see 2.2.1.2) were identified by Veloso [93]. I have arranged these strategies into a taxonomy, as shown in figure 2.7.

Serial replay reuses only a single case. The simplest form of multiple case reuse is *sequential replay*. The retrieval step establishes an order for the retrieved cases; they are replayed in their entirety in that order and then the different plans are connected together. Interleaved replay involves steps from plans being mixed together. In ordering based interleaved replay, plans are played until a "step ordering commitment" occurs. For example, a step in a plan may require a goal; if another step deletes this goal then this imposes a constraint on ordering of the two steps.

Choice and ordering based interleaved replay is similar to ordering based interleaved replay, except that it can use state information to make decisions. An example was given in [93] where a step in a plan (which was being created) added a value to the state of that plan. Then, the plan encountered a goal which could be satisfied by four possible "operators" (plan steps). Three operators had preconditions that were not satisfied in the plan. However, one operator had the value (that was added previously) as its precondition, since this precondition was satisfied then this operator was the obvious choice.

Conceivably, there are different ways in which cases can be combined in design also. For example, if cases A and B are both a reasonable match for the query case, then it is possible that these cases will be split in two, and the parts that form a good match will be merged. However, the merging could be more complex than this. If the designs are hierarchical and the problem can be decomposed recursively, then case A may be reused but some parts of it replaced with case B; however those parts may themselves contain smaller parts of other cases, with merging being applied on different levels of the hierarchy.

Most of the examples here have come from planning or design. Single case retrieval is often inadequate in these complex tasks, and therefore the system developer is left with several choices. One option is to augment CBR with another technology (hybrid systems are discussed in section 2.3). However, the reuse of multiple cases is clearly an attractive option since it mirrors humans typical problem solving strategies (as does CBR in general). People often blend together several past experiences when solving problems. Good examples of this are presented in [94], for biologically inspired design tasks.

Many of the issues that case merging entails can be solved pragmatically according to the needs of particular tasks. For example, if retrieval of a single case is virtually unlikely then it can be discounted and case merging can be considered the norm. However, if interactions between parts of cases are strong, and single case retrieval is likely, then this simpler option can still be allowed for as an alternative within a case-merging system.

Some CBR systems store cases as part of a network or hierarchy (this is particularly true of early CBR systems), with integrity being maintained by links between parts. An alternative is to store the cases as a whole, and use some decomposition algorithm to select the parts. Again this can be determined on a task-by-task basis: if the parts are rarely reused then the better option will be to store the cases as a whole. However, if extensive reuse and retrieval is difficult (e.g. if nearest neighbour algorithms perform poorly) then the links between the parts could form a useful aid to retrieval.

The size of the parts that are retrieved can also be decided on a task-by-task, or case-by-case basis: the number of cases that are retrieved does not have to be constant. In some situations this decision will be obvious, e.g. if a design consists of clearly defined components with minimal interaction then those components will be the parts retrieved. If cases are stored in parts, then it is probably better to make those parts as small as possible, since an algorithm need not be restricted to retrieving just one part of a case, and a fine granularity affords maximum choice. For example in CELIA [45], it is common for several "snippets" of one case to be combined with several from another.

In my opinion, the biggest potential disadvantage of case merging is incompatibility between retrieved parts. The parts may be good solutions to the sub-goals that they address, but there is no guarantee that an integrated solution can be formulated from them. It is possible that a repair activity will be necessary, to facilitate the merging of the parts. For example, the repairs may involve altering the structure of design components so that they fit together, or inserting extra steps into plans so that pre-conditions will be met. But such a repair strategy has no guarantee of success; if structure is altered then the functionality of the components may be affected and meeting the pre-conditions of one part may cause those of another to be violated. However, for some tasks it may be easier to manage these inconsistencies than to have to cope with large differences between the retrieved and query case, which will presumably result in extensive adaptation being required.

Finally, I believe that one potential advantage of case merging is that it could be greatly beneficial where there are *heterogeneous cases*. Heterogeneous cases may reside within the same case base, or different case bases [95]. The latter is referred to as *multiple case based reasoning*, and typically requires a mechanism such as an ontology, since the different case bases may have different semantics. An example of an issue which may arise with heterogeneous cases is: *how can a specification for an apartment be compared to the design of a house?* The differences may not simply be quantitative (the apartment is smaller) but could be qualitative (e.g. the apartment lacks a garage and has less rooms). However, both have a kitchen and bathroom, and it is feasible that any one kitchen can be compared to another regardless of whether it is sited in a house, apartment or commercial office.

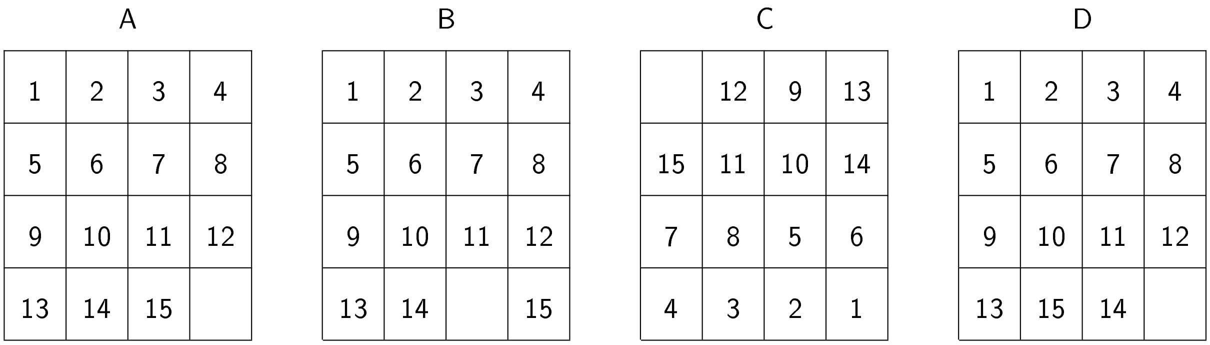
There have been several CBR systems using case merging, and there is work specifically devoted to it in the case based planning literature (e.g. [93]). However, there are very few publications about multiple case reuse in case based design; one of the aims of this thesis is to fill that gap.

2.2.3.4 Failure of Adaptation

For some tasks, there will be an objective way of evaluating the output of the CBR, which the system can measure itself. Thus, the CBR system will be able to detect if it has failed. Then, a number of options are possible. The system can report to the user that adaptation has failed and it is then done manually (null adaptation). Alternatively, it can switch to case merging (see above), or to a non-CBR methodology.

The situation is more serious if there is no way for the system to evaluate its own output. Retrieval only CBR tends to have a fairly predictable output, and it is a concept that is easily explained to non-computer scientists. Retrieval only systems can often justify their output by estimating the degree to which the query and retrieved cases match, which can be invaluable in gaining the confidence of stakeholders (e.g. clients or users). In contrast, adaptation sometimes makes use of complex heuristics and repair strategies, which can make it more of a “black box” to the end user.

Clavier [96] was an interesting example of a CBR system which designed the layout of aerospace materials in autoclave ovens at Lockheed. The materials being heated were of very high value, and the physical properties of the autoclave oven were not fully understood. Adaptation was performed using a mixture of case-merging and heuristics which incorporated domain knowledge. Clavier incorporated a graphical editor so users were able to manually add new cases, assist with retrieval by confirming the



A

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

B

1	2	3	4
5	6	7	8
9	10	11	12
13	14		15

C

	12	9	13
15	11	10	14
7	8	5	6
4	3	2	1

D

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

Figure 2.8: 15 puzzle

choice of retrieved case, and assist with adaptation by manually adapting cases.

When Clavier was evaluated, the users found that the adaptation was not reliable enough, due to the interaction between parts in the oven. For example objects can interfere with convection currents which affect the temperature to which nearby objects are heated. Therefore, *the users did not use the automatic adaptation*, as they had too little confidence in it. Experts tend to be naturally resistant to being replaced by machines, and this resistance may be strengthened if the output of those machines is inferior to their own.

In Clavier, adaptation failed because there was an underlying physical model which was poorly understood. However, it is possible for failure to occur even in well understood domains. Some tasks are undecidable, unsolvable or computationally intractable. Also, there is an underlying assumption in CBR that retrieval distance correlates to adaptation distance. However, in general there is no guarantee that one object can be transformed into another, even if they appear similar. A good example of this is the classic 15-puzzle [14, 97], as shown in figure 2.8. The puzzle consists of a board with sixteen spaces on it, and fifteen tiles. The only legal move is to slide a tile horizontally or vertically into the blank space. There are many variants of the puzzle but the one of interest here has the tiles numbered 1 to 15, with the goal being to achieve layout A in figure 2.8.

We could regard the distance between layouts as the minimum number of legal moves required to effect that transformation from one layout to another. Layouts B, C and D are to be transformed into layout A. Intuitively, B and D look similar to A, but C looks very different. In fact B *is* similar, with just one move being required. C is very different, the distance being 80 moves [98] which is the maximum number of legal moves required to effect any transformation. Layout D, however, *cannot be transformed into layout A* no matter how many moves are made [99].

The example of the 15 puzzle illustrates that, unless in the rare situation in which the solution space is completely understood and adaptation is proven to always be possible, the failure of adaptation must be allowed for. Of course, there are various techniques for minimising this possibility such as adaptation guided retrieval (see section 2.2.3.2). Perhaps the possibility of failure is why so many CBR systems omit adaptation completely.

2.2.3.5 Discussion

Adaptation is the most controversial area of CBR: some researchers [100] believe that deep adaptation nullifies one of the main advantages of CBR, which is the avoidance of the knowledge-elicitation bottleneck (see section 2.4.3). However, this view is not universally accepted, and other researchers consider that retrieval only CBR systems (which are sometimes termed as "case based retrieval" or "CBR-Lite") avoid the more difficult, real-world problems. The consensus in the CBR community seems to be that adaptation has a tendency to rely more on domain knowledge than retrieval. My opinion is that, in complex tasks (e.g. design or planning) where case coverage is sparse, CBR with deep adaptation can serve as a useful compromise between the knowledge-intensive approach of rule-based systems, and the knowledge-light approaches which tend to shift the burden on to the user, who may otherwise have to perform adaptation manually.

This section has explained how the basic adaptation operators can be combined in a variety of ways to make a number of different adaptation strategies. Some of these strategies involve techniques not traditionally associated with CBR (e.g. evolutionary algorithms), so they are in effect hybrid systems. These strategies differ in the amount of adaptation knowledge which they incorporate: as with other aspects of CBR there are knowledge heavy and knowledge light approaches. Case merging is an alternative to adapting a single case which is popular in tasks such as design and planning.

Other systems shy away from adaptation altogether, perhaps because it is somewhat controversial, or maybe because it is difficult to implement and subject to failure. It is also possible that rather than being a complete failure, adaptation will produce results which are mediocre. This can have an impact on the quality of the case base, a topic which is discussed in the next section.

2.2.4 Ensuring quality and efficiency

Once built, some CBR systems continue to perform acceptably without any special intervention. However, other systems have their output affected by the related issues of quality and efficiency. If the content of the case base is causing these issues to occur then this may mean that the system cannot be allowed to acquire cases and grow without intervention; in addition to the processes in the 4REs model, *case base maintenance* may be required.

The drop in *efficiency* arises from the utility problem, as discussed in section 2.2.2.1. As cases are added to the case base, coverage of the problem space becomes denser, therefore the mean distance between query case and retrieved case decreases. It is one of the assumptions of CBR that retrieval distance correlates to adaptation distance, so therefore adaptation distance tends to decrease, which hopefully results in a more accurate output. However, the decrease in retrieval distance tends to become

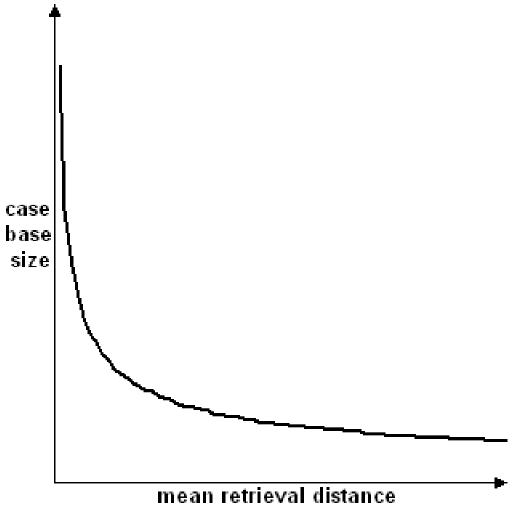


Figure 2.9: Qualitative illustration of the utility problem

gradually smaller, as shown qualitatively in figure 2.9. Therefore, any improvement in adaptation accuracy also tends to become more marginal as the case base gets larger.

Depending on the nature of the retrieval function, increases in case base size tend to result in increased time taken for retrieval, since the retrieval algorithm has a larger case base to search. If retrieval uses a decision tree then the execution times may be logarithmic with respect to the case base size, whereas if every case is examined then it will be linear. So in some CBR systems, a point is reached when adding more cases results in a negligible improvement in accuracy, but an unacceptable deterioration in efficiency. Case based maintenance ensures that execution speed remains acceptable by activities such as controlling the size of the case base.

Efficiency is not the only issue, however. Problems of *quality* occur when there is an inaccurate mapping between problem and solution space. For example, the bold diagonal line in figure 2.10 indicates a problem-solution mapping which clearly appears different from the other mappings. In fact, the problems that CBR systems deal with tend to be multidimensional and not just 2-dimensional. If a mapping is complex then this is acceptable as long as it accurately reflects the reality of the problem domain. However, if it is not accurate[17] then this is likely to lead to quality problems since there is an underlying assumption (see section 2.4.1) in CBR that problems *recur*, i.e. the past can be used as a guide to the future.

In some domains, the nature of the task changes over time. This presents a challenge since older cases may become irrelevant or misleading, which can have an impact on both quality and efficiency. Watson presented [101] an algorithm for case based maintenance in a CBR system which was used for the specification of heating, ventilation and air conditioning (HVAC) systems. The system acquired approximately 5000 cases a year, which were stored in an SQL database. During this growth process,

[17] The concept of an inaccurate problem-solution mapping is not equivalent to the problem being a statistical outlier. For some tasks it may be particularly important for the system to deal with unusual problems effectively, particularly if the task is safety-critical.

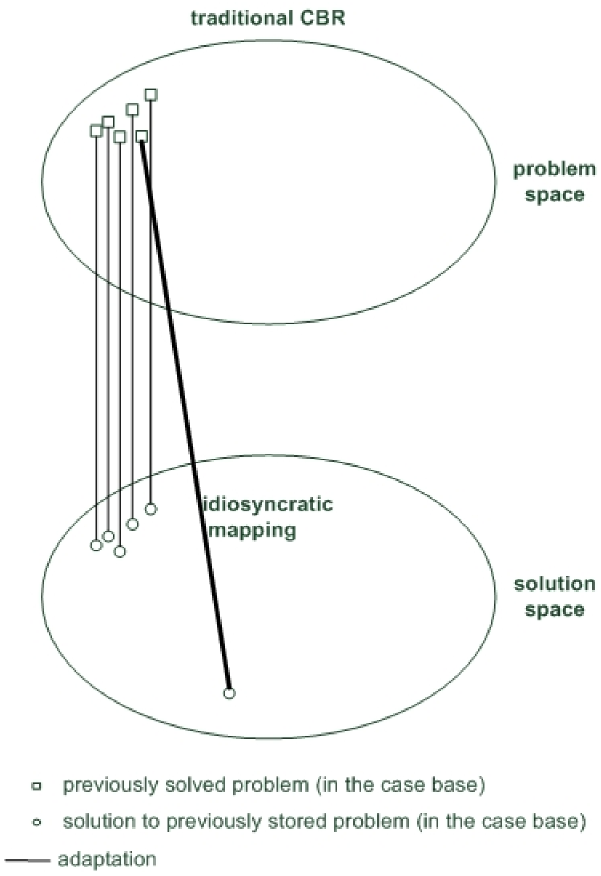

traditional CBR
problem space
idiosyncratic mapping
solution space
□ previously solved problem (in the case base)
○ solution to previously stored problem (in the case base)
— adaptation

Figure 2.10: Idiosyncratic mapping (inspired by [4])

redundant cases were acquired, and some cases became obsolete since they related to discontinued products. All cases had to be stored for commercial reasons, so the cases were assigned a status flag which controlled whether they were suitable for retrieval.

Each evening, when the system was not in use, a case based maintenance algorithm was invoked. The algorithm took all pairs of cases in the case base and computed their similarity. If the similarity of two cases exceeded a preset value, then they became members of the same *similarity set*. Watson's similarity set concept appears to be a specialisation of the mathematical idea of an *equivalence class*. Once all the comparisons had been made, the algorithm found a representative case for each similarity set; this was the one which had the greatest similarity to all cases in the set. Finally, all the members of each set except the representative case were flagged as unsuitable for retrieval. Thus, the effective size of the case base was minimised by the process, which ensured that redundancy was removed automatically. Obsolescence was removed manually with the aid of specially crafted SQL queries.

It is common for academic published works about CBR (and AI in general) to relegate implementation issues as being of minor importance, compared to theory. However, case based maintenance is a very practical topic. Watson's work included no empirical data about performance, so the reader is left unable to judge whether the case-based maintenance activities that he describes are actually necessary. Another disadvantage of the work described in [101] is the crude simplicity of the approach, e.g. comparing all pairs of cases is computationally expensive. Additionally, the algorithm effectively clustered some of the cases, then removed all the outermost elements of each cluster. However, if the cluster happens to be on the edge of a gap, then some of those elements may be valuable in providing coverage for that gap, so their removal is questionable.

A more systematic treatment of case redundancy is given in [102]. The *coverage* of a case was defined as the set of problems that it is capable of solving. The *reachability* of a problem is the set of cases that can be used to solve it. They also defined several categories of case, which are listed in increasing order of importance:

1 An *auxiliary case* can be removed without affecting performance.

2 A *support case* is a type of spanning case (see below) which exists in a group (of other support cases) called a "support group". Individual support cases can be deleted without affecting performance, but if the support group as a whole is deleted then performance will decline.

3 A *spanning case* does not directly affect performance. However the coverage of a spanning case links regions of the case base which are covered by other cases; if these cases are deleted then the spanning case may become important.

4 A *pivotal case* is one which cannot be deleted from the case base without adversely affecting performance.

The goal of the "footprint deletion policy" was to maximise competence whilst minimising the size of the case base. The algorithm deleted cases, prioritising the ones of lesser importance. Within a category, cases can be prioritised by choosing the one with the largest reachable set, or with the least coverage (see above for definitions).

Whilst the theoretical contribution of [102] is interesting, there are real shortcomings in this article. For example, the concepts of coverage and reachability (which underpin much of the paper) are binary; either a case is in the set, or it is not. This doesn't reflect the reality of many problems, which is that set membership really ought to be fuzzy. If a set of cases provides an acceptable solution to a problem then within that set some cases within the set will typically be better than others.

A theoretical framework for case based maintenance was presented by Wilson and Leake in [103]. *Introspective* and *non-introspective* approaches to analysing data were defined. Non-introspective approaches either involve no case-based maintenance, or they use information external to the case base to do it. Introspective approaches were divided into *synchrotic* (which uses a single snapshot of the case base) and *diachronic* (which analyses changes in the case base over time). Diachronic analysis could be useful in a domain where the problem is dynamic, such as in technology or fashion.

Different options for *timing* were also defined in [103]. If case based maintenance is performed at a set stage in the CBR cycle, this is known as *periodic*. Periodic timing can either be *continuous* (if it happens every cycle), or conditional (if it happens in response to a condition, e.g. case base size). In contrast, *ad hoc* timing is defined as being unconnected to the cycle. This terminology is questionable since maintenance that is performed every evening would be termed ad hoc, which seems counter-intuitive. Similar to timing, two modes of *integration* were defined: *online* (which is during the CBR cycle) or *offline*. Offline could refer to time when the system is idle between cases, or when it is waiting for user input. It appears that there is some overlap between the concepts of timing and integration.

Options for triggering case based maintenance were also defined. Maintenance can be invoked when the case base reaches a certain size (*space-based*), when retrieval times exceed a threshold (*time-based*), or when the system fails to give good results (*result-based*). Time-based and result-based will normally be the most relevant, since storage in modern computer systems is often relatively cheap. Time-based maintenance addresses mainly efficiency problems, whereas result-based focusses on quality problems. Alternatively (rather than being *reactive*) maintenance can be *proactive*, where problems in the future are predicted. In general, extrapolation and predicting the future are activities that are notoriously hard to do well. However, an example is given in [103] where a company is planning to release a range of new products: this might change the nature of problems in the case base so it could be a good time to do case base maintenance.

The scope of case based maintenance is categorised in [103] into operations which have a *narrow* scope (e.g. affecting one or two cases), and a *broad* scope (e.g. changing the indexing structure). There are three levels that can be affected by case-base maintenance: *implementation* level (e.g. changing indices), *representation* level (e.g. changing the names of features) and *knowledge* level (e.g. deleting cases). If the nature of problems change, it is possible that the nature of the features will change, e.g. an integer value might need to become a real value, this would be at the representation level.

Wilson and Leake's framework shows that there exist a large variety of options for implementing case-based maintenance. Case-based maintenance has traditionally been seen as being about removing cases from the case-base, however there are alternatives such as making use of indices. If performance is a problem, and decision trees are not part of the retrieval strategy, then introducing indexes ought to be considered, as they can lead to substantial improvements in efficiency, without having to remove any cases.

Both indexing and removing cases were discussed in [104], and are described as *eager learning*. In eager learning, the "training data" (i.e. case base for CBR) is used to craft the learning mechanism in advance of the problem. In contrast, CBR is normally considered to be *lazy learning*, since the actual reasoning only happens when the problem is input into the system. Experimental evidence in [104] has shown that implementing a case based maintenance strategy can result in the case base having reduced coverage, and the system having greater errors in its output.

The experiments performed in [104] were quite simple, they generated random data with a Euclidean distance measure. In real life, data can be complex, with interactions, redundancy, conditional behaviour and non-linearity. Whilst it may be possible to construct a distance function f, it may be difficult to find the inverse function f^{-1}. Finding the distance from the query case to a solved case might be easy, but finding the area of the case base that is "covered" by that solved case could be much more difficult. So, caution must be exercised if cases are to be removed based on calculations of their "coverage".

A separate but related issue to case based maintenance is that of *seeding the case base* for new CBR systems. CBR systems need a case base to work! There are two options for dealing with this issue. Firstly, a system can be started with an empty case base, but only if it is a hybrid system, and if the complementary technology is capable of providing solutions with no training data. Presumably, if training data was available, this would be loaded into the system as cases in advance of the system "going live", which is the second option. If the system is a pure CBR system (i.e. not a hybrid), this second option may be the only choice; the system cannot be used until it is pre-loaded with cases. In some situations it may be possible to seed the case base with cases manually created by users or imported from another computer system. But not just any cases will do; they need to be representative of the sort of problems that the system is presented with. They need to provide sufficient coverage so that a typical query case will have a nearest neighbour in the case base that is suitable for retrieval. If there are large gaps in the coverage, then the new CBR system will begin its life with quality problems.

In my opinion, the decision as to whether to employ case-based maintenance for efficiency should be a pragmatic one, based on knowledge of the retrieval algorithm and empirical data. CBR is not often used for applications where the output is fed into another program or automatic process, such as device control. Typically, CBR systems are employed in tasks where a human user is the consumer of their output. So, the threshold for what is an acceptable level of efficiency is determined by the user. If the execution time is of the order of milliseconds, then this will appear instantaneous to a human. If the rate of growth of that execution time is sufficiently small, then it may be several years before any case base maintenance is required. Also, the extent to which computing power will develop in that time-frame is not certain. Computer systems usually have a finite life; they are often discontinued because they are superseded with something better, or because the problem they solve is no longer relevant. So, in some cases the system will end its life before efficiency problems arise, or those efficiency problems will be cancelled out by improvements in the performance of computer hardware. Rather than predicting that case base maintenance will be required, it may be better to wait until problems arise:

> "We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil" - Donald Knuth [105]

> "More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason - including blind stupidity" - William A Wulf [106]

Case base maintenance for quality is a different issue. If products have become obsolete, for example, then their corresponding cases may still be able to provide valuable lessons. It may be possible to craft the retrieval algorithm so that obsolete products are not retrieved unless they can make a particularly valuable contribution to a new problem; for example, perhaps the circumstances of the old and new cases are both unusual but similar to each other.

The problem with poor solutions being stored in the case base is a more serious one. Algorithms exist within the machine learning community to remove redundant and noisy data; for example jCOLIBRI comes supplied with such algorithms (see section 2.2.5.1). However, as with all knowledge light methods, this would only be appropriate if the dimensionality of the problems is small enough compared to the size of the case base.

Another option is to let experts remove these solutions, as in HOMER [8] (see 2.2.1.1 for more detail). However, this may be an expensive and labour intensive process. An alternative is to store a measure of success with the case, and this can be used to guide retrieval. In some problem domains, objective measures of success are available, however in others it is simply a matter of asking the user to rate the solution, e.g. on a scale of 1 to 10. Presumably, the poorer quality solutions would only be retrieved if they happened to be particularly similar to a new case, and no good quality solutions were applicable.

Building a case-based reasoning system can be something of an onerous task. Case based maintenance, if required, will be just another activity that adds to the time taken to develop the system. However, as the next section shows, short-cuts to developing a CBR system are available.

2.2.5 CBR Shells

Case based reasoning systems can often be complex and large software systems, which are rich in functionality. All large software projects carry with them risk and cost, and so it makes sense for a system developer to look at ways to reduce these. Although CBR systems work in many different problem domains, they all follow the same principles. They all have a case base, they all have retrieval and retention, and some have adaptation also. Given this common ground, it is unsurprising that *CBR shells* have been produced: these offer a generic framework which a developer can utilise to quickly create a full CBR solution.

2.2.5.1 jCOLIBRI

jCOLIBRI (http://gaia.fdi.ucm.es/projects/jcolibri) is by far the most successful CBR shell, and it is the subject of many publications, e.g. [107, 108, 109, 110]. jCOLIBRI is implemented as a Java program with associated XML files, and a graphical user interface (GUI) editor. The XML files describe tasks that are involved in the CBR process, e.g. retrieval. These files must adhere to a standard schema, which is supplied. The developer is free to add tasks to the XML, but each task needs to have an associated method, which must be implemented in Java.

The Java package has abstract classes and interfaces which correspond to all the important processes and structures in CBR, such as Case and SimilarityFunction. jCOLIBRI is provided with some default concrete classes which extend the abstract classes or implement the interfaces, providing default functionality. The developer is free to use these classes, or to write their own in order to provide different functionality.

jCOLIBRI provides a means of organising the cases into a case base: the interface between persistent storage and volatile in-memory storage is referred to as a "connector". Connectors are provided for cases stored in XML files, and in relational databases. To assist with maintaining the case base, some algorithms are included for removing cases which are redundant or "noisy", i.e. cases which give the wrong classification. A number of problem solving methods are provided to perform key tasks such as retrieval; for example, there are several in-built similarity functions. A case visualisation tool allows the developer to check the appropriateness of the similarity function. There is also a means to perform adaptation by using rules.

The GUI provides a rapid, easy means of configuring the tasks in jCOLIBRI. A tree shows how the tasks involved in CBR are decomposed in various levels. The topmost level has four high-level tasks: Retrieve, Reuse, Revise and Retain. Lower level tasks relate to specific methods and the choice of methods can be changed by the user using the interface. The connectors can also be configured in the GUI, for example it can be used to map attributes.

jCOLIBRI contains several advanced features, such as the ability to interface with ontologies such as OWL, and a tool which facilitates the development of web-based interfaces. There is also limited support for adaptation in jCOLIBRI, where the case representation uses an ontology [110]. Developers can store adaptation rules in text files; the rules have three parts. The first part identifies the "instance" (attribute of the case) to be adapted; the second part is a condition that must be true for adaptation to occur; and the third part is the definition of the adaptation. Substitution is the only supported adaptation mechanism; the value to substitute can be specified in the text file or obtained using relationships in the ontology.

Whilst jCOLIBRI's support for adaptation is welcome, it is perhaps disappointing that it is only available to CBR systems which use an ontology. However, since jCOLIBRI allows the developer control over which tasks it performs, it would be possible for adaptation to be incorporated as Java code. This gives the developer almost complete freedom over the adaptation mechanism; for example they are not restricted just to substitution.

The most impressive aspect of jCOLIBRI is how it can be utilised by developers with differing levels of expertise: beginners can concentrate of configuring the GUI, whereas advanced developers can extend the algorithms by writing their own program code.

2.2.5.2 myCBR

myCBR (http://mycbr-project.net) is GUI-driven, and its key advantage is cited as being the ability to do rapid prototyping [111]. myCBR focusses on retrieval, the goal was to enable a developer to create a small CBR system with a complex similarity measure, with minimal effort being required. myCBR is based on the ontology editor Protégé; both myCBR and Protégé are open source. Whilst the CBR system is being developed, myCBR runs as a "plug-in" within Protégé. There is a GUI similarity editor which can be used to build weighted sum type similarity measures. A data import tool can import data from CSV files into an existing ontology, or new one can be created.

Similarity measures are defined on both local and global levels; the latter is an aggregation of the former. The global measures are typically Euclidean or weighted sum. The local measures can be more sophisticated, e.g. trigram matching for strings, taxonomies for symbolic values, and mathematical functions for continuous numeric values. A particularly innovative feature of myCBR is the explanation capability, which can justify the system's output.

The many limitations of myCBR are apparent: it does not handle adaptation, and the case base is restricted to being stored in an ontology. However, it remains a useful tool for building small prototype retrieval-only systems.

2.2.5.3 Other tools

The **AIAI CBR Shell** (http://www.aiai.ed.ac.uk/project/cbr/CBRDistrib) is a tool which concentrates on retrieval, with a weighted sum approach. An interesting feature is the ability to optimise weights using a genetic algorithm. The tool can be ran as a Java applet within a web browser, meaning no software installation is required.

CBR*Tools (http://www-sop.inria.fr/axis/cbrtools/usermanual-eng) is an object-oriented framework for building CBR tools [112]. It consists of a set of Java classes; some of these are abstract and are designed to be subclassed by the CBR system developer. Others are concrete and implement useful functionality, e.g. k-d tree indexing.

Lastly, one particularly innovative tool has been dubbed **"CBR for CBR"**. jCOLIBRI was extended, resulting in a recommender CBR system which produces prototype CBR systems [113]. They defined a *template* as a series of tasks; one or more templates were retrieved and adapted to make a CBR system. A library of templates was built, which the developers could select from, for example "single shot system". The "single shot system" template was decomposed into tasks such as "one-off preference elicitation".

A case in the system was a CBR system, stored with its templates. They used an interactive method that they termed *retrieval by trying*; this simply meant that the user is given an opportunity to run the retrieved CBR system to check if it is suitable or not. The similarity measure compared the methods that were used in the CBR systems; these were arranged in an ontology. There are several applicable similarity measures when ontologies are used in CBR [110], e.g. computational based retrieval; this uses recursion over the ontological hierarchy to match values, applying a weighted sum to each. The interactive nature the system described in [113] is interesting but there are disadvantages to relying on the user to validate outputs; eventually people become fatigued and tend to pay less attention to the result.

In general, the emphasis of CBR shells tends to be on retrieval, so these tools may provide a useful short-cut for research into retrieval algorithms. jCOLIBRI is the best supported and most functional tool, which even includes some support for adaptation. All tools have their limitations and disadvantages. One universal disadvantage is the time taken to learn how to use the tool, and to understand its capabilities and limitations. For example, the *introductory* user guide to jCOLIBRI is 47 pages long [114].

If a shell is only GUI based, then this is an obvious limitation since many bespoke algorithms would be impossible to specify in a GUI. If it involves programming, through a mechanism such as abstract classes, then there are more subtle limitations: the programmer will have to implement the abstract methods of those classes. They will not have the (almost) total freedom over the structure of their software that comes with programming a project from scratch.

Whether it is advisable to use a CBR shell depends entirely on the circumstances. CBR shells are an effective way to build prototype systems or academic demonstrator systems with minimal effort. However, if the features of the CBR task are particularly complex and specialist, then they are less advantageous. If adaptation is needed, then jCOLIBRI is the only applicable choice. However, jCOLIBRI's support for adaptation is very limited so if adaptation is complex and a large part of the CBR system, then it may not be appropriate.

In general, if extensive adaptation is required, and the cases are very complex and require a sophisticated used interface, then CBR shells are unlikely to be a good choice. In these situations the benefits that the shell brings may be minimal, and a competent software engineer may be able to construct such minimal functionality fairly quickly anyway.

2.3 Hybrid Systems

A *hybrid CBR system* is any computer system where CBR is combined with another problem-solving methodology. Hybrid systems are also known as *multi-modal reasoning systems* [115]. Such systems are often very successful since the two methodologies can compensate for each others weaknesses. Several hybrid systems have been discussed so far in this chapter, e.g., in section 2.2.3.2. This section first

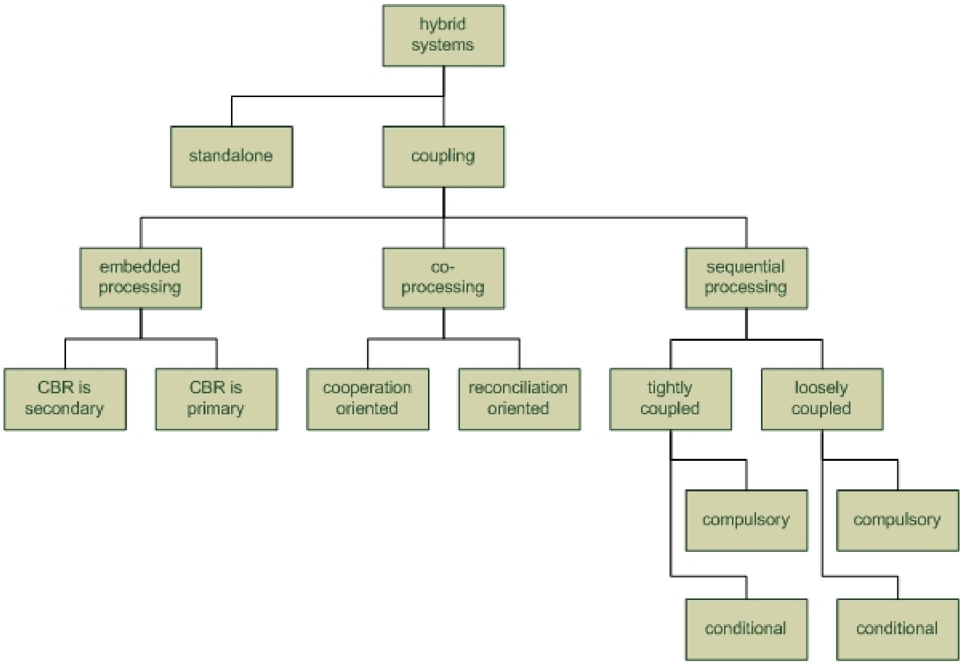


Figure 2.11: Taxonomy of hybrid systems, adapted from [5]

discusses the nature of hybrid systems in general. This is followed by a brief survey of other methodologies which can be hybridised with CBR.

Prentzas and Hatzilygeroudis [5] devised a classification scheme for combining rule-based and case-based reasoning, as shown in figure 2.11. In fact, the scheme could apply to systems in which CBR is hybridised with other methodologies, and it is used in this more general way in the following discussions.

The *standalone* classification refers to situations in which the two systems do not interact at all. The user receives the outputs from the two separate sub-systems and utilises them as they wish. For the standalone approach to work, both CBR and the other methodology have to be able to handle the problem as a whole. In many cases they cannot (or cannot handle them well), which is why the standalone approach is rare. The other assumption is that the user is able to do something with two sets of outputs, which may contain areas of agreement and disagreement. One possible use for the standalone approach is testing the applicability of CBR versus other methodologies for a particular task.

Hybrid systems with *coupling* involve some form of interaction between the two systems. The simplest sort of coupling is *embedded processing*, in which one of the two methodologies is the primary problem solver, but it has the other methodology embedded within it. The responsibility of the secondary methodology is to perform a sub-task for the primary methodology. For example, a rule based reasoning system might be embedded within CBR, with the responsibility of performing adaptation. CBR can play either role: sometimes the task may lend itself to the CBR paradigm, but one of the steps in the CBR cycle may require the assistance of the other methodology to implement it. Alternatively, CBR may be used to implement a sub-task in some other problem solving methodology.

Watson argued that CBR is a methodology and not a technology [41]. Since CBR can be implemented in a variety of ways, Watson felt that using rules for adaptation was just one of many implementation options for CBR, and should not be considered a form of hybridism [41]. In this thesis, a more relaxed view is taken; such systems can often incorporate the best of both methodologies, so they are worthy of discussion here.

In *sequential processing*, the two sub-systems are invoked one after the other, in such a way that one system has finished before the other has started. Sequential processing systems can be further categorised along two dimensions:

- Degree of coupling: In a *loosely coupled* system, the output of one subsystem does not play a major role in the reasoning processes of the other. Alternatively, in a*tightly coupled* system, the information from one sub-system is vital to the other.
- Conditionality: In a *compulsory sequence*, one sub-system is always invoked after the other. However if the sequence is *conditional*, then the latter sub-system is not necessarily invoked; the decision may depend on the success of the first sub-system.

Sequential processing can be useful in situations where the success of one methodology cannot be guaranteed. For example, model based reasoning can be attempted, but if it fails then the system can resort to CBR, in a conditional sequence. Compulsory sequences could be useful if one methodology has to do some form of pre-processing, for the benefit of another. For example, CBR could be employed first to generate solutions which are close to being correct, but not perfect; those solutions can then be 'tweaked' with the other methodology. In a medical domain, CBR might be used to provide the diagnosis, then some other methodology could provide the treatment.

Co-processing is the most complex form of hybridism. In co-processing the sub-systems interact and information flow is bidirectional; the different sub-systems may work in parallel or be interleaved in some way. In *cooperation-oriented* approaches, the different sub-systems work together to provide the solution. In contrast, *reconciliation-oriented* approaches involve the sub-systems working more independently; the results are compared at the end and then the system reconciles them to provide one answer. Co-processing could be complex to implement, particularly if tasks are interleaved; one system may have to wait for another. Reconciliation oriented sequences may be advantageous if the task is a safety critical one; if the answers are very similar then a high degree of confidence can be attributed to them since they came from independent sources.

2.3.1 Rule-based reasoning (RBR)

Rule-based reasoning was the first methodology to be hybridised with CBR [32, 115]. The synergy often works particularly well since rules represent general knowledge, whereas cases represent specific knowledge. If a problem domain contains both types of knowledge but neither CBR nor RBR can solve the problem on its own, then a hybrid system is a natural choice. For example, many early CBR-RBR systems were in legal domains [32]. Within a legal domain, a system can model laws as rules, and precedents as cases. Cases can be used to provide a flexible interpretation to rules, such as capturing exceptions that are not represented in the rule base.

GYMEL [116] was a CBR-RBR system which specialised in harmonising musical melodies. Rules existed for this domain, and these were taught to students in music schools; however the rules were more like guidelines than certainties. It was thought that to implement GYMEL as a pure CBR system would require an unrealistically large number of cases. So, a hybrid system was the natural choice.

> “the rules don’t make the music, (it) is the music which makes the rules” - Sabater et al [116]

When a new problem was presented to GYMEL, it first attempted a solution using CBR. If the CBR failed (because a similar case could not be found), the task was given to the RBR module. The output of the RBR was then given to the CBR module to see if improvements could be effected. GYMEL was classified by Prentzas and Hatzilygeroudis [5] as a co-operation oriented co-processing system.

2.3.2 Model-based reasoning (MBR)

MBR works by encoding an underlying *strong theory* problem domain. The first CBR-MBR system [32] was CASEY [20, 21], which was discussed in section 2.1.2.2. CASEY was primarily a CBR system, which relied on a MBR module to provide output in case the CBR failed. Using CBR was more computationally efficient than using MBR, hence a hybrid system was appropriate. CASEY was a sequential processing system with a conditional sequence in the classification scheme of Prentzas and Hatzilygeroudis [5]. In general, possible reasons for wanting to augment MBR with CBR are:

1 The model is incomplete - it only covers part of the problem domain.

2 The model is only an approximation.

3 The model requires excessive computation or is intractable.

In the case of CASEY and FormTool (see section 2.1.2.3), option 3 was applicable. In fact Kolodner cites one of the advantages of CBR as being that solutions can be retrieved quickly rather than being laboriously worked out “from scratch” [57]. However, this is not universally true: sometimes derivation from first principles may be the most efficient option.

2.3.3 Evolutionary Algorithms

Evolutionary algorithms were briefly discussed in section 2.2.3.2; Evolutionary algorithms begin with a set of arbitrary solutions, which are then modified and combined by a stochastic process. Not all solutions are retained and the process favours good solutions, in an analogy with the *survival of the fittest* in natural selection. Genetic algorithms (GA) is a sub-field of evolutionary algorithms, and GA and CBR have been successfully combined in the past.

GENCAD [84] was a CBR system which used GA for adaptation; its name derives from GENetic Case ADaptation. GA was also used to learn the weights for a weighted sum retrieval algorithm in [50], as discussed in section 2.2.1.2. In the classification scheme of Prentzas and Hatzilygeroudis [5], these are both embedded systems, where the CBR is the primary methodology.

CBR tends to have a predictable output, often deterministic. In domains such as design, sometimes some randomness or creativity is an advantage. However, creativity has to have its limitations; the solution must fit the problem in hand. Thus, CBR-GA systems combine the advantages of creativity the reuse of past experience.

2.3.4 Others

Constraint satisfaction problems (CSP) have been hybridised with CBR ([85]; see section 2.2.3.2). The constraint satisfaction algorithms were involved in adaptation, by facilitating case merging via managing inconsistencies between the several cases that were received. Thus this system was an embedded system, with CBR as the primary methodology [5]. Where constraints can be represented formally, and adaptation is complex, a CBR-CSP system may prove effective.

Information retrieval (IR) refers to the use of statistical techniques which use carefully crafted queries to retrieve data from large collections of data [32]. SPIRE [117] was an example of a CBR-IR system which retrieved text from legal documents. SPIRE worked by first retrieving documents from the case base. Then, the most relevant documents were passed to a query engine, which would automatically generate a query from terms in the documents. The query would then be run against a larger corpus of text, and a set of documents returned. SPIRE was then able to form a query to locate passages of text from that set of documents. Those passages were conceptually related to a term that is relevant to the user's query, e.g. "sincerity". SPIRE contained a case base of excerpts of text that were related to particular terms, and it retrieved relevant excerpts then used them to formulate a query. This query could then isolate relevant packages from the set of documents. SPIRE was an example of a co-operation oriented co-processing system [5]. The advantage of SPIRE was that it was able to utilise powerful querying technology, without requiring the user to explicitly formulate those queries.

CBR has been combined with *fuzzy logic* in numerous ways; typically the fuzzy logic is used to enhance the CBR system [118]. For example in FormTool [29], a "fuzzy preference function" was used in retrieval. Fuzzy logic could be useful in domains where there is uncertainty and imprecision.

CBR has been combined with *ontologies* in the generation of stories [119]. The domain knowledge was represented in an ontology using OWL. Tests were done with several different similarity measures which were incorporated into *CBROnto*, an ontology which comes supplied with CBR concepts [120]. For example, similarity can be based on the most specific concept which subsumes two particular cases, known as the *least common subsumer*. Ontologies are useful when concepts from the problem domain naturally lend themselves to being stored in a hierarchy.

A CBR system which incorporated both fuzzy logic and *neural networks* was used to forecast potentially harmful blooms of phytoplankton in oceans [121]. A case consisted of a small vector of numeric values, including environmental data and the concentrations of the plankton. A neural network was used in retrieval. Four fuzzy systems were used for adaptation; each subsystem was assigned a weight. When the actual values for the plankton concentrations were received, these were used to update the weights of the subsystems, so that the most accurate ones were given priority. This is a good example of a CBR system that incorporates more than two methodologies. Thus, the taxonomy in figure 2.11 is a simplification, albeit a useful one.

2.4 Analysis of CBR's capabilities

This chapter has so far defined and explained the concept of CBR, explored the options for constructing a CBR system in detail, and discussed the issues behind hybrid CBR systems. Although hybrid CBR systems have their place, they illustrate that CBR is just one option amongst many other methodologies. The suitability of CBR for tasks (in general) has not yet been discussed. Many methodologies which are far more established; CBR began in the 1980s (see section 2.1.2.1), but artificial neural networks were starting to be developed as early as 1954 [122]. So*, why use CBR, when so many other techniques are available?* This section addresses this question by examining the assumptions, strengths and weaknesses of CBR.

Many academic works present CBR in an uncritical light. In some ways this enthusiasm is unsurprising because historically some of the problems that CBR is good at solving were intractable by any other means. However, this section provides an overview and objective and impartial discussion of the assumptions behind CBR, and its advantages and disadvantages. It is necessary to be familiar with these ideas in order to make an informed decision as to whether CBR alone will be suitable for a particular problem, or whether the solution is a hybrid system or some other methodology altogether.

2.4.1 Assumptions of CBR

The CBR model relies upon several assumptions for it to work. Some CBR systems assume that adaptation is possible, and this assumption cannot always be guaranteed to hold (see section 2.2.3.4). Also, as with any methodology, CBR systems may make additional assumptions based on their problem domain. However the assumptions listed below lie at the core of the CBR model.

The past is a guide to the future

The most basic assumption of CBR is that the information stored in the case base will be of use to solve problems in the future. This means that, to some extent:

- *Situations are repeatable*: CBR may not be appropriate, for example, for medical diagnosis in a specialist hospital dealing with rare diseases. Some diseases are so rare that they only affect two people in the whole world [123].

- *What worked in the past will work today* : The design of antibacterial drugs may be a task unsuited to CBR. Bacteria are capable of mutation and become resistant to drugs, so agents that work in the past will not necessarily work today. CBR may sometimes be used in changing problem domains, provided extra care is taken in maintaining the case base (see section 2.2.4).

Similar problems have similar solutions

Another basic assumption is there is some correlation between the difference between any two problems and their solutions. If the mapping between problems and solutions is described as a mathematical function, then that function ought to be continuous. For example, CBR would be an unreliable predictor of values of the signum function (see below) due to the discontinuity around x=0.

$$sgn(x) = \begin{cases} -1 & if\, x < 0 \\ 0 & if\, x = 0 \\ 1 & if\, x > 0 \end{cases}$$

Similarity is quantifiable

Given a query case Q, CBR assumes that:

- For any case A in the case base, the similarity function $sim(Q, A)$ is computable, *or*

- For any two cases C and D in the case base, it is possible to compute which of C or D is more similar to Q.

In some domains, e.g. art, it is very difficult to objectively quantify the similarity of any two objects. This, however, is an issue for any AI technique, and not just CBR.

2.4.2 Some definitions about domain theory

As with any technique, the advantages and disadvantages of CBR cannot be considered in isolation of the sort of problems that it is typically suitable for. So, some definitions about domain theories are presented here, from [124]. A *weak theory* domain is one in which there is substantial uncertainty associated with the relationships between terms. In contrast, a strong theory domain is one where the relationship between terms is more certain; if there is complete certainty then it is said to be *perfect*. A *tractable* domain is one where an efficient algorithm exists to computer the terms. The *inverse* concept of this is an *intractable* domain. Examples of all cases are:

1 Finding the zeroes of a quadratic equation is a *tractable*, *strong theory* problem. A method for solving quadratic equations was recorded as early as 628AD [125]; it is exact (perfect theory) and easy (tractable).

2 The traditional oriental game of *go* is an *intractable*, *strong theory* problem. The definition of a winning position in *go* is completely unambiguous, as are the rules about what constitutes a valid move, so it is perfect theory. The number of distinct board positions is approximately 10^{170}, and approximately 1.2% of these are legal [126], so the size of the search space makes it intractable.

3 Computing a weighted sum for similarity is a *tractable*, *weak theory* problem. The underlying domain may contain ambiguity, making it weak theory; so the weighted sum algorithm is just a heuristic. However, the use of a weighted sum tends to be computationally very efficient, since it usually consists of simple arithmetic.

4 Choosing layouts for autoclave curing, as in Clavier [96], is an *intractable* and *weak theory* problem. Although certain heuristics about the domain were known, the effects of changes were unpredictable, so it was weak theory; the number of possible layouts was large so it was intractable. For more information about Clavier [96], see section 2.2.3.4.

2.4.3 Advantages of CBR

CBR has been applied to many problem domains, due to its inherent advantages.

Often works in poorly understood problem domains

CBR often works in weak theory domains (see above for definition), where there are no reliable algorithms for solving the problem are available [57]. In some ways, CBR mimics the “common sense” approach that humans have to solving problems. Humans typically solve far more problems in their daily lives using common sense, heuristics and experience than they do through formal logic or mathematical methods.

Does not normally require a large knowledge engineering exercise

Techniques such as model based reasoning and rule based reasoning typically require an extensive knowledge acquisition exercise. This may involve many interviews with experts, with the associated tasks of

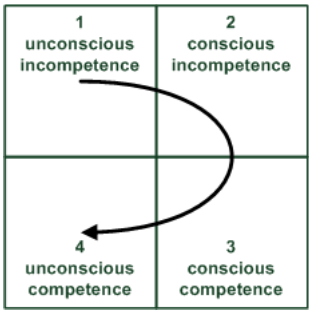


Figure 2.12: 4 stage competence model

recording and analysing the findings. This can be a lengthy and time consuming task, and it is sometimes only partially successful. The domain experts that are interviewed tend to be at stage 4 in the widely cited 4 stage model of competence[18] shown in figure 2.12:

1 Learners start off as an *unconscious incompetent.* They are unaware of their own limitations: they don't know what they don't know!

2 The next level of progression to *conscious incompetent*, where the learner becomes aware of their limitations, but has not yet overcome them.

3 At the *conscious competent* stage, the learner knows how to do things, because they have recently acquired skills, and they are aware that they are using those skills.

4 The final stage is *unconscious competence*: the learner is now an expert, who performs tasks quickly and effectively using instinct, habit and wisdom, rather than explicit rules or logic.

So, when experts are interviewed by the knowledge engineer, they often struggle to articulate *how* they do their job [2], since they are not consciously aware of (at least some of) their decision making processes. They might be able to describe heuristics which solve the more straightforward problems, but explaining the special cases could prove more difficult. And experts often make assumptions about what the interviewer knows, which are not always accurate. Acquiring *tacit knowledge* is notoriously difficult and unreliable. This is known as the *knowledge elicitation bottleneck*.

A key advantage of CBR is that the amount of knowledge acquisition is typically much less than with rule-based (or model-based) systems. In CBR, the knowledge is in the cases themselves, rather than being explicitly encoded into the system. Note that there is an important caveat here. In several places in this chapter, knowledge light and knowledge heavy have been used to describe contrasting approaches to some aspect of CBR, e.g. retrieval or adaptation. So, I am not claiming that developing a CBR system involves no knowledge acquisition, just that the level of knowledge acquisition is typically less than with some other approaches.

[18] This model is of disputed origin, although Noel Burch is quoted as its originator [127].

Gradual degradation of performance

As a consequence of the assumption *similar problems have similar solutions*, the performance of CBR degrades gradually as problems get further away from that portion of the problem space that is covered by the case base. This means that if a CBR system is presented with something which is an outlier, the results may be poor but they will probably not be disastrous. In contrast, rule-based systems [1] and model-based systems [9] tend to be *brittle*; they are notorious for having a narrow, fixed area of expertise, and for performing badly when those limited boundaries are challenged.

Ease of maintenance

Some CBR systems need very little maintenance: if the nature of the problem changes over time, the nature of the case base tends to change, and this is reflected in the output. So, although some maintenance activities may be required (see section 2.2.4), these are not normally too onerous and may consist simply of removing some older cases. There is an element of dynamism built into the CBR paradigm: as long as a new case can be stored using the case representation, it can be entered into the case base and it will change the nature of the case base in line with the changes in the problem. In contrast, rule-based systems can become more and more difficult to maintain as they expand [2].

Ability to offer intuitive explanations

For classification tasks, the ability of a system to offer an explanation can be very important in gaining the confidence of users. In section 2.2.3.2, CASEY [20, 21] was discussed; this was a hybrid CBR/model-based reasoning system which worked within the domain of heart failure, and was able to offer explanations of its output.

The explanation of a CBR system would be able to refer to the case that was retrieved. The idea of solving problems based on problems in the past is not an inherently alien or complex one to most people, so a carefully crafted explanation is likely to be accepted as “common sense”. In contrast, other AI techniques are often not able to offer as intuitive an explanation as CBR can. Although rule-based systems can offer a rule trace, if this was long it could be incomprehensible to a non-technical user. A model based system might be able to provide a statistical or mathematical justification of the output, but again this could be opaque to a non-expert user. Neural networks have very poor explanation capabilities; they are traditionally regarded as a “black box”. So, the ability to offer easy-to-understand explanations is a definite advantage of CBR over other methodologies.

Can incorporate domain knowledge

Several sub-goals in the CBR process (e.g. retrieval) have alternative methods of achieving them which can be described as knowledge heavy, or knowledge light. Each has their respective merits and drawbacks: knowledge heavy approaches can involve an onerous knowledge acquisition task, partly defeating one of the main advantages of CBR. However, sometimes knowledge heavy approaches are the only ones that work; if domain knowledge is available, should it not be made use of? At least the knowledge heavy options are available in CBR. Some methodologies such as neural networks cannot be easily augmented with domain knowledge. It is possible for a CBR system to take have the benefit of past experience, domain knowledge, and techniques outside of the CBR paradigm.

2.4.4 Disadvantages of CBR

As well as its many advantages CBR has some disadvantages, as do all methodologies.

Case based reasoning requires cases

CBR needs an adequate coverage of the case base in order for it to give acceptable performance. Tasks where cases are not available, for example a one-off computation where no previous data exists, are therefore unsuited to CBR. This would be an issue for many other AI methodologies; technologies such as neural networks, support vector machines and decision trees all require training data.

Cases must be stored

CBR requires on-demand access to a case-base to work. Storage in modern computer systems is relatively cheap, so nowadays this is rarely a problem.

Solutions are not necessarily optimal

Case based reasoning is a *heuristic* technique. It tends to return good solutions but it cannot normally be guaranteed to return the optimal solution. Consider the example problems in section 2.4.2. In weak theory domains (options 3 and 4), there is probably no precise definition as to what constitutes the optimal solution, so the fact that solutions may be sub-optimal is not necessarily a problem. Tractable, strong theory problems (1) would not be suitable for CBR, since by definition there exist far more straightforward algorithmic methods.

Some intractable, strong theory domains (2) would also be unsuitable for CBR. For example, CBR would be an inappropriate methodology for an automated *go* game player. This is because even if a previously won game was found with a similar position to the current one, it would not necessarily help the search for an optimal move.

Other problems in category (2) might be suitable for CBR, because they could use previous cases to initialise a search procedure. FormTool [29, 30], which was discussed in section 2.1.2.3, is an example. However, CBR is still not guaranteed to give perfect answers in this sort of problem, since there may be gaps in the coverage of the case base. When a query case is presented to the system, if it resides inside the gap area, then it might not be practical to compute an exact solution. This is because the problem in the retrieved case will be quite distant from that in the query case, and so we assume the solutions will be distant. Intractable domains have wide search spaces and so the search might take an unacceptably long time. So, the system must either report that it has failed, or output the best possible approximation obtained within an acceptable time.

Fortunately, in the case of FormTool, the optimal solution was not necessary; it was a *satisficing problem*. Also, it is worth noting that the problem in FormTool was strong theory, but not perfect theory; the domain model was an approximation. Hence the users of FormTool were satisfied with very good solutions, without requiring optimal ones.

Generating complex objects can be difficult

Synthesis tasks involving complex objects, are notoriously difficult for any methodology. In the case of CBR the difficulties are twofold:

- Complex objects are multi dimensional, and therefore it is difficult to get good coverage of the case base.
- Adaptation is difficult, since by definition it is difficult to transform one complex object into another.

To some extent, these difficulties are interchangeable. Richter [128] viewed CBR as consisting of *knowledge containers*; deficiencies in one area (container) can be made up for by the others. Thus, adaptation is used to remedy gaps in the case base.

If coverage is poor or adaptation is intractable, then the CBR system developer ought to ensure that easier, alternative methods of generating the objects do not exist. However, it is often the case that CBR is employed because other approaches are not applicable or have failed.

This chapter has shown CBR to be an effective problem solving methodology; it works using an intuitive model which is said to mimic human problem solving. It has been used since the early 1980s across a wide range of tasks and problem domains. The many different ways to build a CBR system have been highlighted; these allow the developer of the system to choose to utilise previous experiences, domain knowledge, and other artificial intelligence techniques in solving the task in hand. This chapter has shown the CBR model to have a lot of flexibility built into it, and this flexibility is undoubtedly the main reason for the success of case based reasoning as a problem solving methodology.

Chapter 3

Design

"The details are not the details. They make the design."- Charles Eames

The design of hand knitting has similarities to any other design process. Before considering how the process can be supported or automated, it is important to consider the nature of design itself. Section 3.1 begins by discussing *design studies*, an interdisciplinary field which has been actively researched since around 1962 [129]. The focus of design studies is on the human aspects of the process of the design activity.[1]

Section 3.2 highlights some recent developments in computer-aided design (CAD). Research in CAD is fundamentally different to design studies, in that it concentrates on tools to automate the design process (rather than the human aspects of that process). Section 3.3 discusses specifically design of knitwear and garments. Finally, section 3.4 divides design (in general) into two categories, and explains the relevance of these categories to the problem in hand.

3.1 Research in Design Studies

Cross [130] asserts that designing is an activity which is inherent to humans. Designers are apparently focussed on the outcomes of design, and are often unwilling or unable to discuss the activities that take place during the design process. Cross points out that this gives design an almost magical air of mystery. Design is an abductive[2] process which uses intuition, draws on the broad life experiences of the designer, and involves spontaneous flashes of inspiration. In order to demystify this process, design researchers use a variety of tools such as interviews, observation, experiments, simulations and reflection/theorising.

In the case studies by Cross, the design journey begins with a *design brief*, which is less well defined

[1] In contrast, architecture and engineering tend to focus on the outcomes of design, and the technical aspects of the design process.

[2] *Abductive reasoning* can be loosely defined as hypothesising or guessing. It contrasts with *deduction* (reasoning from general rules to a specific proof, such as in mathematics) and *induction* (reasoning from specific observations to a general rule or theory, such as in science).

than a specification, meaning that the designer has freedom to be creative. Some designers take pride in the fact that their goal is to produce a design that is not necessarily what the client asks for, but is something that exceeds their expectations. Problems are often poorly structured and it is said that the problem to be solved is not necessarily the problem that was given.

Since a human's short term memory is limited, they are likely to use sketches as a design aid. Cross refers to traditional pencil and paper drawing, since this allows designers to be spontaneous, and switch instantaneously between different levels of detail. They may utilise *emergent features*, such as shapes that unexpectedly appear when components are overlaid on each other (this is a form of interaction between design components).

Designers are characterised by Cross as people who thrive on uncertainty and risk, and who work on ill-defined problems. They often jump to a solution quickly, before the problem is fully formulated.They work on parallel lines of thought, but adopt a systems approach to their work, i.e. they take a holistic view of design.

Cross talks of the design process as being like a journey, starting with the design brief. The designer is apparently often not able to see the whole route of the journey, but just their immediate surroundings. Thus they may appear (to an outsider) to take a indirect route to their destination. Novices apparently use a depth first approach to design problems, whereas an expert will use a mixture of breadth first and depth first. Cross mentions that designers often prefer to work from first principles, rather than using the innovations of others. Designers are described as being motivated and competitive.

Some of the ideas discussed in Cross's work [130] are intuitive. For example, the inability of designers to discuss their processes is possibly because those processes are underpinned by tacit knowledge. Design is a practical activity and much of design education has a large practical component; some connect the methods of learning by experience to the ubiquity of tacit knowledge [131].

Other ideas are debatable, e.g. the assertion that designers prefer not to reuse others innovations. This is perhaps unsurprising if expert designers work in a competitive culture. However, design is derivative at some level, even if the designer is not consciously aware of it. Another design guru, Larry Leifer, has formulated a series of design laws, one of which is "all design is redesign" [132].

The fact that apparent contradictions (such as the one discussed above) exist within design studies research is unsurprising given that it uses theorising and reflection as part of its methodology [130]. Critics may accuse design studies as lacking rigour. However, in fairness, the inner workings of a designer's mind cannot be observed objectively; this is also true of some other fields, e.g. cognitive science [7]. The irony is that perhaps both design and design studies can be said to use inspiration, intuition and abductive reasoning.

3.2 Computer aided design (CAD)

Since CAD tools are well established (for example Autodesk [133] was developed in 1982), CAD is a mature discipline and the basic concepts are well-established. Current research focusses on special topics as discussed below.

Curves

The representation of curves continues to be an active research area. For example, non-uniform rational basis splines (NURBS) are used to represent curves [134]; these are generalisations of Bézier curves [135]. NURBS offer flexibility and precision, and can be made to fit point clouds using interpolation and approximation techniques [136].

Representation and visualisation

In addition to the work on curves, there is active research in visualisation, meshing models (e.g. using quadratic surfaces [137]), the representation of boundaries, and relief extraction. 3D-CAD has been established for some time, but innovations continue; for example, methods have been used to generate 3D drawings from 2D ones which are are assumed to be "orthographic parallel projections". Various optimisations have been found as the process is computationally intensive [138].

Manufacturing

Some CAD work is on the interface with engineering. For example, techniques are available to simulate milling, turning and drilling [139]. Such simulations are rapid and cost-effective; they allow the parameters of the engineering process to be fine-tuned without manufacturing real prototypes. Other work has concentrated on the use of optimisation techniques to reduce the errors in machining [140].

Other work

The above categories are broad but by no means all-encompassing; much work is done in CAD that is interdisciplinary in nature. For example, a knowledge-based CAD system called DANE uses a Structure-Behaviour-Function model to facilitate biologically inspired design [141]. Also, techniques have been developed to use text mining to capture design rationale (the reasoning behind design choices) [142]; thus the reuse of design knowledge is facilitated.

In general, CAD research focusses on engineering problems, where shapes are complex and sophisticated mathematical models are used. Typically, a high level of precision is required, and the costs of error are high. Nevertheless many of the generic issues in design (as discussed in section 3.1) apply. For example, accurate visualisation is paramount in order to support the designer's creativity.

3.3 Knitwear and garment design

Eckert is the most prolific researcher in machine-produced knitwear design, mainly approaching the subject from an interdisciplinary design studies point of view. Eckert has authored or co-authored over 26 publications with relevance to knitwear design between 1994 [143] and 2006 [144], and completed one of the few PhDs in the subject [145].

Eckert undertook a series of extensive studies into the garment industry. Early findings included the divided nature of the personnel involved [143, 145]. The earlier activities in the design process are undertaken by designers, and the latter ones by technicians. The designers generate the specifications, and the technicians use CAD software to turn these into reality. However, communication difficulties arise between the two groups of people, and the specifications are often "inaccurate, incomplete and inconsistent" [145]. The technicians often complained that the designer's specifications were infeasible, and the designers complained that the patterns produced by the technicians had deviated from their specifications to an unacceptable level.

In order to overcome these difficulties, Eckert proposed a system for specifying garment shapes. "Mock-ups" constructed using a drawing package were provided to show the suggested appearance of such an interface [145]. The mock-ups show a set of points connected by lines and curves. Eckert developed several mathematical models of garment shapes, mostly based on (quartic and quintic) Bézier curves. In hand knitting, there is a lesser requirement for precision than machine knitting (see section 4.7), and so quintic Bézier curves would probably be unnecessary: they have the drawback that four points need to be specified.

The models suggested enable curves to be represented with a fairly high degree of precision. Eckert was concerned with ambiguity as well as imprecision [145]. A system of notation was proposed to reduce the ambiguity that is present in freehand sketches.

Eckert also conducted research into the reuse of designs. There is a discussion of how creativity is often measured by the uniqueness of a design [146]; however there is also analysis of the adaptation of previous designs. A designer's sources of inspiration are vital to the creative process and the utilisation of those sources is termed "adaptation". This term is also used in case-based reasoning (see section 2.2.3), which is briefly mentioned by Eckert as having utility in design problems [145, 146]. It seems likely that there is extensive reuse in design, if not of the designs themselves but the sources of inspiration; therefore it is no surprise that case based reasoning is often cited as being an appropriate methodology to support design. As well as case-based reasoning, Eckert briefly discusses the use of evolutionary algorithms in design [145, 146].

The use of genetic programming to generate lace knitting stitch patterns was investigated by Ekárt [147]. The pattern was represented in a novel way, as an ordered set of trees, to ensure it remained technically valid. The biggest weakness of evolutionary algorithms is the need for a fitness function; in this work, automatic evaluation was performed using some simple metrics. Evolutionary algorithms can produce some surprising results, and thus are arguably capable of creativity. However, this creativity could be a disadvantage if the purpose of the system was to produce designs according to the specific goal of a user.

A system to enable members of the public to design their own garments was developed by Woodford [148]. Due to the novice nature of the users, the goal of the system was to provide advice and assistance to them. Woodford mentions case based reasoning as an option but states that the fundamental characteristic of his situation was that the user did not have an initial goal in mind, other than to simply design a garment. Their goal was developed and refined as a result of using the system.

Similarly, work has been done on customisation of machine produced knitwear [149]. This is a slightly different problem (to those discussed by Eckert or Woodford) since what is required is a co-design tool, where the customer can specify their requirements, with a little help from a shop assistant. The assistant will first show them samples of yarn and patterns. They then design the garment together and the design is sent automatically to the knitwear CAD system at the factory without time-consuming manual programming being required. The authors present this achievement as a “breakthrough”, which is illustrative perhaps of the complexity of commercial knitwear CAD systems, or maybe of the difficulty in technically interfacing them with other systems.

There is much research in knitwear that is not specifically related to the design process itself, such as work in the commercial aspects of the garment industry, 3-D modelling of garments, the physical properties of yarns, the development of new yarns, ubiquitous computing (integration of technology into textiles), and the use of knitwear in specific domains such as medicine or sport. However, these subjects are outside the scope of this thesis.

3.4 Categorising design processes: creative versus mechanistic

Much of the discussion in section 3.1 refers to design processes that are highly creative. Design is ubiquitous in developed societies, and it seems intuitive that some of this design must be more routine in nature. So, I distinguish between *creative design* and *mechanistic design*, as per table 3.1. In the studies performed by Eckert (described in section 3.3), creative design is done by designers and mechanistic design by technicians. In the processes at Sirdar Spinning Ltd (see section 5.1.1.1), the initial “design” stage is creative design, whereas the other stages are mechanistic design.

Since mechanistic design is less creative, the question may arise as to whether it can be termed “design” at all. Indeed, at Sirdar Spinning Ltd the term “design” is reserved for only the initial stages. However, I believe that mechanistic design can be considered a design process, since creative and mechanistic design

Creative design	*Mechanistic design*
begins with a high-level design brief	begins with a detailed specification
the problem to be solved is not necessarily that which was given	the problem solved is identical to or has a close relationship with the one given
works on ill-defined and poorly structured problems	works on well-defined and well-structured problems
uses a variety of sources of inspiration, including abstract ones such as the designer's own memories	sources of inspiration are usually predetermined, obvious or straightforward
design process is idiosyncratic, using intuition and flashes of inspiration: it is not predictable, even to the designer	design process is mostly predictable
significant uncertainty and risk	limited uncertainty and risk
mostly abductive reasoning	mix of abductive and deductive reasoning
emphasis on novelty, creativity and general aspects of functionality	emphasis on correctness and detailed aspects of functionality

Table 3.1: Creative versus mechanistic design

have common aspects. For example they both depend on expertise, and that expertise often relies on tacit knowledge. Designers are often not consciously aware of how they select and adapt their sources. At Sirdar Spinning Ltd, a pattern writer appeared not to be consciously aware of the rule they used to avoid a ribbed cuff tightening (see section 4.6). They asserted that there was no rule, whereas in fact they added almost exactly 17% more stitches on each occasion.

Both types of design involve constraints, although perhaps in creative design these are more likely to be soft constraints: the designer is given more freedom to relax them. Also, both creative and mechanistic design involve the possibility for inconsistency, inaccuracy and incompleteness. The "high-level" sketches drawn by the designers at Sirdar Spinning Ltd are prone to ambiguity as they are not drawn to scale and not every feature will have its measurement specified. At the mechanistic level, I have personally seen detailed sketches of garments drawn by pattern writers which have inconsistent measurements on them.

Freehand sketching and CAD packages can be utilised by either type of design. However, it seems likely that freehand sketching will be more important in creative design, as it allows for spontaneity and flexibility. Conversely, CAD will be more useful in mechanistic design, if the package supports the process by ensuring compliance with the specification. Emergent features can occur in either type of design; the interaction between features can be utilised to benefit the design. For example, a curve may be specified in a piece of fabric which is to be knitted in a lace pattern, and the pattern can be altered in a natural way, to facilitate the shaping that makes the curve.

Reuse can be a feature of either type of design. In creative design, both the design and the sources of inspiration can be reused. In mechanistic design, both the designs and the way in which they were derived can be reused. The obstacles to reuse are that in mechanistic design, the designers may refer to explicit rules which they use in their work; there is no need to reuse specific designs if good general rules are available. However, it is worth noting that the exceptions to those rules are not always articulated. In creative design, the chief obstacle to reuse is the designer's pride: they see each design as being individual, and a suggestion that it is in any way derivative is not always met with a positive reaction.

When discussing the differences and similarities between creative and mechanistic design, it is more helpful to think of them as being extremes on a spectrum, rather than a dichotomy. The pattern writing at Sirdar Spinning Ltd is mechanistic design, but the pattern writers exercise creativity in their work, albeit more limited than the "designers".

One aim of this PhD is to support the latter, mechanistic stages of design. The initial stages are best done using freehand sketches, using the designer's existing tools such as mood boards. The latter stages can be supported by a system which provides a visual record of the designer's work with prompt feedback, automates rules and constraint checking, and facilitates both creativity and the reuse of creativity. Chapter 5 explains how a CAD system was developed to support mechanistic design of knitwear. The reuse of creativity is facilitated by the case-based reasoning functionality that is explained in chapter 7.

Chapter 4

The Knitwear Domain

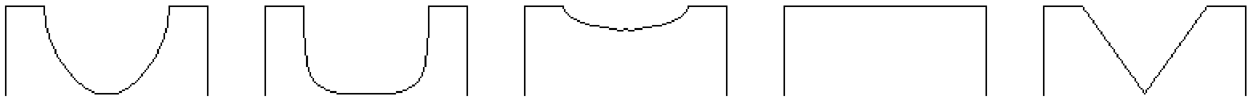
> "I will resist the urge to underestimate the complexity of knitting." - Stephanie Pearl-McPhee (2005)

Hand knitting is a leisure activity in which a human knitter produces garments, typically by following a series of textual instructions (a *knitting pattern*), previously produced by knitwear designers. The fabric that composes the garments is made from *yarn*, which is a long string made from natural or synthetic fibres (e.g. wool, nylon or blends of these). The knitter manipulates the yarn with tools known as *knitting needles* to make stitches. *Stitches* are loops in the yarn that are connected to each other. Each type of stitch has a different effect on the shape or appearance of the fabric that is formed.

Knitting patterns are typically several pages long and written in a codified language that is often incomprehensible to a non-knitter. Pattern design is a complex process which has artistic, technical and commercial considerations. The finished garment must be fashionable and aesthetically pleasing, as well as wearable. The design process produces artefacts at different level of detail, culminating in the finished knitting pattern. The designer has a choice of many knitting stitches, and there are technical and artistic constraints on how these stitches are used to achieve the desired goal.

Constraints exist in nearly all design processes, and so are not unique to knitwear. However, what is striking about professional knitwear design is the scarcity of literature. There are books for knitters, and amateur knitwear designers, but the heuristics used in professional knitwear design are largely undocumented. Although each design is unique, there is an element of repetitiveness to the process. Knitwear designers employ a lot of tacit knowledge in their work.

Little academic study has been undertaken in hand knitwear design. However, the inherent complexity of knitting, and the partially documented nature of the design process with its many constraints, arguably makes this domain an interesting subject for research. This chapter summarises the domain knowledge necessary to undertake research into the case based design of hand knitting.



(a) Round neck (b) Scoop neck (c) Slash neck (d) Straight neck (e) V neck

Figure 4.1: Neck shape options

4.1 Designing sweaters and cardigans

This dissertation concerns the special case of designing of hand-knitted *sweaters* and *cardigans*. These are both types of garments that are worn on the torso, and optionally they can have sleeves to cover all or part of the arms of the wearer. *Sweaters* are also known as pullovers, because the wearer will put them on by pulling them over their head. *Cardigans* are similar to sweaters, the difference being that they are divided at the front. Hand-knitted cardigans are usually (but not always) buttoned at the front.

Hand-knitted sweaters and cardigans are usually composed of individual *pieces* of knitting, which are knitted separately and then sewn together afterwards[1]:

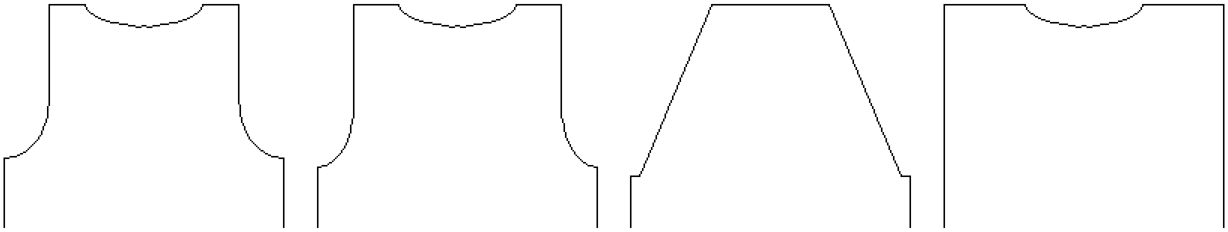
- Body pieces such as the *front* and *back* are always present. They are roughly rectangular in shape, with exceptions for the neck and armhole, as described in section 4.2. Also, if the garment has a *fitted* or *baggy waist*, then the basic shape will be closer to an irregular hexagon than a rectangle.
- Two *sleeves*; although some garments are sleeveless. The shape of a sleeve loosely approximates to an elongated trapezium. The widest part of the sleeve is known as the *head* and is discussed in section 4.2.
- An optional *collar* or *hood*.

Although most garments can be knitted (e.g. socks, hats or gloves), this study is restricted to cardigans and sweaters,partly because these are the most common types of garment encountered in hand knitting. More importantly, the design of cardigans and sweaters is suitable for automation as it tends to follow a common set of principles common principles: it is an example of *variant design*.

4.2 Shapes used

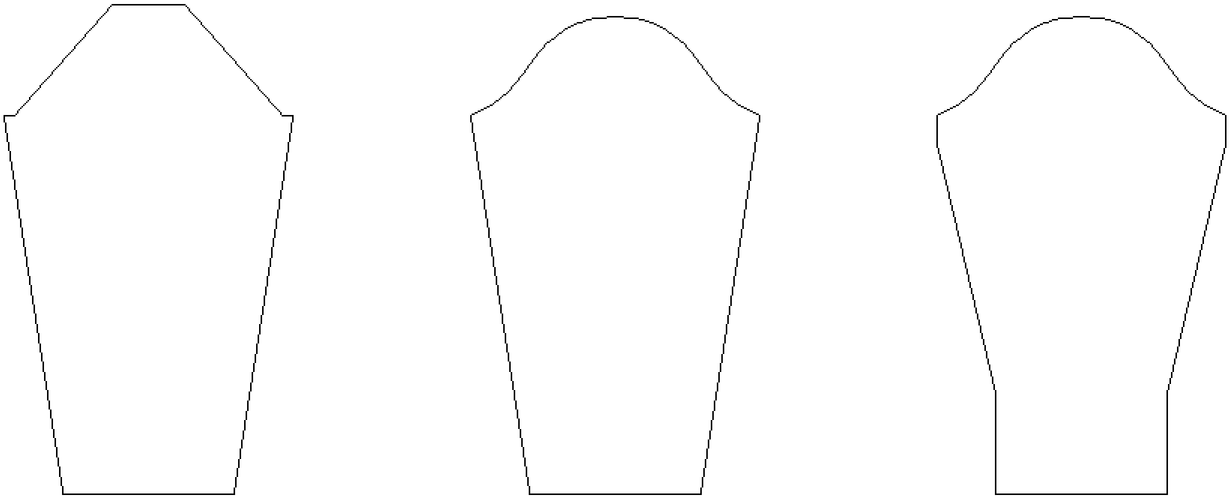
When a knitwear designer wants to turn their artistic inspiration into a practical, wearable and fashionable garment, they give careful consideration to the shapes of the pieces. Whilst most shapes are knittable, in practice there are are commonly chosen options for most features. Since there is not universal agreement on nomenclature, the terms for those options will be defined here.

[1] It is possible to use other techniques such as knitting sideways or circular needles to avoid having as many separate pieces as this; however this is not the norm and so is excluded from our consideration.



(a) Set-in armhole (b) Semi set-in armhole (c) Raglan armhole (d) Dropped shoulder

Figure 4.2: Armhole shape options (with a slash neck)



(a) Raglan sleeve (b) Non-Raglan sleeve (c) Non-Raglan sleeve with straight portions

Figure 4.3: Sleeve shapes

Figure 4.1 shows examples of the common options for the *neck shape*. Some garments have a greater or lesser neck depth than others, and a designer may choose a shape which is a variant on these. Nevertheless, for example, if a round neck was stretched vertically to make it more like an ellipsoid, it would still be called as a round neck. The shapes involving curved lines, i.e. round, scoop and slash neck could be described or specified by Bézier curves, whereas the more simple shapes involve straight lines only. Note that the back and front of a garment do not necessarily have the same neck shape. Indeed it is most common for the back to have a slash neck.

Figure 4.2 shows the common *armhole shape* options. The armhole is located at the top of the body pieces and allows a gap for the arms to fit through. The curves used in the set-in and semi set-in armholes can be described adequately by an elliptical quadrant.

In a sleeved garment, the head of the sleeves will be connected to the armhole by the pieces being sewn together. When the *Raglan*[2] style is used, the front almost always has a V-neck, for aesthetic reasons.

The neck and armhole shapes both relate to the body pieces. However, the shape of the head of the

[2] Raglan sleeves were named after Lord Raglan [150].

sleeve is normally a variant of two styles; the one shown in figure 4.3a is used with a Raglan armhole, and that in figure 4.3b with armholes of other styles. The shape of the non-Raglan sleeve head may vary qualitatively, but it can generally be described by a Bézier curve.

Figure 4.3c also shows a non-Raglan sleeve, with the head identical to 4.3b. However the difference is that figure 4.3c has two additional horizontal lines which are referred to as “straight at bottom of sleeve” and “straight at top of sleeve”. The former is usually for reasons of style, whereas the latter is to make it easier to knit. These two features are optional and some garments have only one of the straight parts.

4.3 Measurements and constraints

The design of hand knitting has aspects in common with other design processes, e.g. in engineering. However, one key difference is in *precision*. No knitting pattern would express a measurement more precisely than to the nearest millimetre. In fact, some measurements are usually to the nearest centimetre for commercial reasons: it makes the pattern easier to understand for the knitter. If the number of stitches is quoted (instead of a measurement), this is rounded to the nearest integer.

Symmetry is a common soft constraint in knitwear design. The sleeves are nearly always symmetrical about a horizontal axis, as shown in figure 4.3. Also, the right and left sleeves are almost always identical when knitted, and mirror images of each other when the garment is assembled. A consequence of this is that patterns usually only specify one sleeve, and state “sleeves both alike”. Also, the back and front are normally symmetrical about a vertical axis. However, asymmetrical garments are technically possible to knit and are sometimes a design feature, in response to particular fashion trends.

Consistency of measurements is the key technical constraint. It is important that, when pieces are sewn together, that the measurements match. However, there is considerable leeway on this, since knitted fabrics are capable of stretching. So, for the measurements to be within ±5% of each other is acceptable. Consistency constraints apply to:

- *Armhole - sleeve head*: the arms and the body pieces must be able to be sewn together. This is probably the constraint that is easiest to get wrong, as the shapes are usually not simply composed of straight lines.
- *Body length*: since the front and back are sewn together on the sides, the lengths must be equal.
- *Shoulder length*: the body pieces are also sewn together at the shoulder.
- *Body width*: the bottom of the front and back is normally the same length, otherwise the seam would not run down the side of the wearer, which could spoil the aesthetics of the garment.
- *Border widths*: if there is a bottom border (see section 4.6) on the front and back, this normally has the same width for aesthetic reasons.

Regardless of the shapes used in them, knitting patterns are not commercially viable unless they come in a range of sizes to suit the intended wearers. Traditionally in the UK the size of a cardigan or sweater was measured in inches, and this represents the circumference of the wearer's torso across the chest or bust. Nowadays the patterns use both metric and imperial for measurements. The size range is normally specified as an arithmetic sequence of inches where the terms are all even numbers. Adjacent to each term is shown the size in centimetres, rounded to the nearest integer[3].

When determining the measurements used in their garments, designers typically use a standard table of measurements. The table is a rough guide to help the designer, and is not usually in the public domain. It will typically list the standard length and width of the front, back and sleeve for each of the possible sizes (e.g. 30"-50"). A smaller (e.g. 34"-42") range of sizes will usually be specified for a garment, since the table represents some extreme scenarios; also the table may be unisex but most knitted garments are not. The designer will pick a size in the range (often the smallest) and specify the measurements with reference to those in the table. They may use the actual measurements, but leeway is possible to suit fashion trends. For example, sometimes baggy garments are popular; these use slightly larger measurements, e.g. "as table width plus 2cm".

In order to derive the measurements for the garments in the other sizes within the range, a process known as *grading* is performed. Grading normally involves scaling the design; the scale factors are usually derived by comparing the guideline lengths and widths from the standard table. Thus, those scale factors may be different for the sleeve and the body pieces. However, grading is not as simple as applying a uniform scaling. The standard tables typically specify constraints for some individual measurements, e.g. the depth of the armhole. So, the designer will apply localised adjustments to these features to comply with the guidelines. In addition, there are other complications to the grading process:

- If the measurements were different from the guideline (in the standard table) in the original size, one would expect that they would also be different in the other sizes, presumably by the same proportion.

- Consistency (see above) must be respected in all sizes.

- Rounding is applied (see above for the comments on precision).

4.4 Stitches

So far, the discussion of knitwear design has been at a relatively high level. However, for most knitters the focus is on the lowest level of detail - the stitches. The number of types of knitting stitch is probably

[3] Hence the rounding means that the measurements in centimetres will not necessarily be an arithmetic sequence.

Category of stitch	*Stitch size*	*Stitches linked to in previous row*
Standard	1	1
Cast on	1	0
Cast off	1	1 (but 0 in next row)
Cables	n (where $n > 1$)	n
Shaping stitches	n (where $n \geq 1$)	$n + d$ (where $d \neq 0$)

Table 4.1: Categories of knitting stitches

finite but very large; however in reality only a few dozen are in common usage. The stitches listed in appendix A cover the majority of knitting patterns.

The knitted fabric is formed by making rows of stitches; within each row the stitches are connected to each other and to the stitches in the previous row. Each type of stitch has a textual abbreviation; these are not completely universal so in knitting patterns there is always a legend which explains how to knit each stitch.

Knitting stitches can be grouped into four categories:

- *Standard* stitches include the most common choices: *knit* and *purl*. Knit and purl make up the majority of most garments.

- *Cast on/off*: to begin a piece of knitting, the knitter will *cast on* stitches, these will form the first row of knitting. On the last row they will *cast off*.

- *Cables*: these are decorative stitches which are larger in size than standard stitches. For example, the stitch C4B is the equivalent of 4 standard stitches.

- *Shaping stitches*: a shaping stitch can be used to change the number of stitches in a row, thus it can make the knitting non-rectangular. Stitches which make the row longer are called *increases*, and ones which narrow it are *decreases*.

Table 4.1 summarises the effect of the categories of knitting stitch on linked rows.

4.5 Stitch Patterns

A *stitch pattern* is a 2-dimensional matrix of stitches which achieves a predetermined aesthetic affect when knitted. The number of possible stitch patterns is virtually infinite, although the vast majority of these would never be used as they would give an ugly appearance or would be difficult to knit. Appendix A lists some common stitch patterns; the most ubiquitous of these is *stocking stitch*. In stocking stitch the rows alternate between knit and purl stitches.

It is possible to categorise the majority of stitch patterns thus:

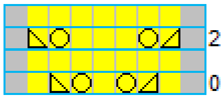


Figure 4.4: Example of a knitting chart showing a stitch pattern

```
1st row (rs): k1, *k1, k2tog, yfwd, k1, yfwd, s1 k1 psso, k1; rep from *, k1
2nd row: purl
3rd row: l1, *k2tog, yfwd, k3, yfwd, s1 k1 psso; repeat from *, k1
4th row: purl
Repeat these four rows.
```

Figure 4.5: Example stitch pattern: as textual instructions

- *Plain patterns*: these give an unremarkable appearance, e.g. stocking stitch or reverse stocking stitch.
- *Ribs*: these are very stretchy and show straight parallel lines in the knitting, where there are ridges and troughs. In contrast to other types of stitch pattern, ribs are specified using two integer parameters. These parameters control the width of the ridges and troughs.
- *Lace*: these involve holes for decorative effect.
- *Cable*: these employ cable stitches, and are often used on the front of sweaters for decorative effect.

Knitting is done in a zigzag or boustrophedonic way. The knitter will usually start on the right-hand side of the piece, and work from right to left. The next row will be knitted on top of the previous one, and will work from left to right, and so on. The rows alternate between the *right side* (which forms the outside of the garment) and the *wrong side* (making the inside). For reference in a knitting chart (see below), the rows are numbered. The following conventions are most commonly observed:

- The right side (rs) is typically worked right to left and has even numbers.
- The wrong side (ws) is typically worked left to right and has odd numbers.

Stitch patterns are often specified using *knitting charts*. A sample knitting chart is shown in figure 4.4. Each of the rectangles corresponds to a knitting stitch which is uniquely identified by a symbol; for example, the circle directs the knitter to make a hole. The symbols are explained in appendix A. Figure 4.5 shows the textual version of the stitch pattern in figure 4.4.

The pattern has the following noteworthy features:

- A blank square on a knitting chart corresponds to stocking stitch. This means that it will be knit (k1) on a right side row, and purl on a wrong side row.

- The columns in grey in the knitting chart correspond to those stitches that are excluded from the horizontal repeating part of the pattern. This is denoted by asterisks (*) in the textual instructions. This means that this pattern repeats horizontally every 7 stitches.
- The pattern repeats vertically every 4 rows.
- The instruction "yfwd" makes a hole in the knitting and it is an increase.
- The instructions "k2 tog" and "s1 k1 psso" are decreases; they are added to balance out the effect of the yfwd (see validity below). They have different aesthetic affects though: the former creates a shape that slants to the right, and the latter to the left.

Stitch patterns have one important technical constraint: *validity*. Often, for the reassurance of the reader, a knitting pattern will quote the count of stitches in a row. As table 4.1 shows, the knitting stitches have a different effect on the *stitch count*. For example, the decrease stitch k2tog ("knit two together") decreases the count by one since it is connected to two stitches below. An increase such as yfwd ("yarn forward") increases the count by one as it is linked to no stitches in the previous row. Cable stitches are not shaping stitches but, for the purposes of calculating the stitch count, they count as more than one.

The validity constraint means that any change in the stitch count from one row to the next must be matched by the cumulative effect of the shaping stitches. In figure 4.4, the stitch count does not change since the increases are balanced out by the decreases.

If the shaping stitches are not balanced out, then the stitch count will change. For example, if the only shaping stitch in a row was k2tog, then it would have a stitch count of one less than the previous row. This is often exploited for deliberate effect, for example to make the fabric narrower for an armhole. It is normal for stitch patterns to have their shaping stitches balanced, so to form the shapes described in section 4.2 (such as a tapered sleeve) the row must be made unbalanced by changing the composition of shaping stitches.

There can be complications if the stitch pattern being used contains shaping stitches. Stitch patterns repeat regularly over the fabric (every 7 stitches and 4 rows in the example above). Often, the number of stitches in a row is changed so that it is an integer multiple of the repeat period (7n+2 stitches in the example above). However, it is not always possible to do that, and if only part of a pattern is used then that may not be balanced and thus could lead to the row being invalid.

The options for changing the composition of shaping stitches include:

- Removing existing shaping stitches.
- Adding new shaping stitches.
- Changing the type of shaping stitch used. Some stitches have a greater or lesser effect on the stitch count than others (see appendix A for details).

As explained above, a change in the stitch count will result in a change in the width of the knitted fabric. Stitches take up an approximately rectangular area of the fabric. The size of a stitch is determined by the *tension*. The width of the rectangle is given by the *stitch tension* and the height by the *row tension*. The tension is affected by four factors:

1 The yarn - the thicker the yarn, the greater the tension.

2 The needle - knitting needles come in various sizes and smaller needles usually result in a lower tension.

3 The stitch pattern - for example, rib patterns have a lower stitch tension than stocking stitch.

4 The style of knitting - some knitters will knit tighter stitches (i.e. a smaller tension) than others.

Manufacturers of yarns normally specify what the tension for a yarn should be, when it is used for stocking stitch with a needle of a specified size. To control the last factor, knitters are encouraged to knit a 10 stitch x 10 row tension square, and measure it to determine whether they are working to the correct tension. It is customary to quote tensions as the number of stitches per 10 centimetres rounded to the nearest integer, for example "22 stitches and 30 rows to 10cm".

4.6 Other design options

Section 4.5 explained how stitch patterns can be specified, and how stitches can be used to shape a garment. A knitting pattern normally specifies a particular pattern as being predominant throughout the garment; we call this the *background stitch*. The background stitch may be the only pattern used on the garment, but if other patterns are used they are specified on a case-by-case basis. Often, additional stitch patterns are used on a garment for aesthetic reasons. For example, they could be used for *edging*, for example:

- *Bottom border*: this affects both the front and back pieces.

- *Front border*: a cardigan (but not a sweater) may have a border running down the front; often the buttons and buttonholes are located on the border.

- *Neck band*: a band is a thin strip of material which is used to "finish off" the neck in the absence of a collar or hood.

- *Cuff*: the sleeves may have a separate region at the end. Sometimes the design specifies that this should be tighter (narrower in width) than the rest of the sleeve, but conversely it can also be looser (wider).

The edging regions are often knitted in a different direction to the rest of the garment; the main portion of the piece of knitting is knitted first, then the edging is added. It is possible (but uncommon) for edging to be done using the background stitch. Usually, the background is stocking stitch, and the edging is done in a rib pattern (this is particularly true of bottom borders and cuffs). Ribs have a lower stitch tension than stocking stitch (which is the dominant stitch pattern in most garments), so the designer has so make allowances for this. For example, if a tight cuff is not required then the cuff will have to contain more stitches per row than the rest of the sleeve.

Pockets are also typically created after the main part of the knitting has been done. They usually have a different stitch pattern to their surrounding area, and are typically rectangular shaped. However, pockets are typically composed of two layers of knitwear superimposed on each other; the top layer is connected to the bottom by sewing or knitting. Pockets are both functional and decorative.

In contrast, panels and yokes are purely decorative. *Panels* are regions which are knitted in a different stitch pattern to their surroundings. These are often rectangular shaped, but can also be a horizontal or vertical band. A *yoke* is a region at the top of the front and back which is knitted in a different stitch pattern to the background stitch.

4.7 Comparison with other processes

Much of what has been presented in this chapter could apply to any method of producing garments; the exceptions are the detail of stitches and stitch patterns (discussed in sections 4.4 and 4.5 respectively). Therefore, it is useful to compare hand knitting to other processes which produce garments.

Knitting normally uses one thread of yarn[4], whereas *weaving* uses many threads. In weaving, vertical threads are arranged on a device called a loom, then separate horizontal threads are interlinked or woven through this. Woven fabric has much less capacity to stretch than knitted fabric; therefore when designing a woven garment there is less leeway in the measurements and a higher level of precision might be expected.

In modern times, most garments are mass-produced using machines. *Machine knitting* has different characteristics to hand knitting. In hand knitting, shaping is always performed with the use of shaping stitches, whereas a machine can knit a rectangle and then the required shape is cut out afterwards [145][5].

Another important (but easy to overlook) factor is that the finished product of commercial hand knitting design is a knitting pattern. There are commercial constraints on knitting patterns, in addition to those of the garment. The pattern must not be too long; a long pattern will be more expensive to print and could be off-putting to the purchaser. Similarly, the complexity of the written instructions must be minimised. If the instructions are too complex then the knitter may become confused and the probability

[4] Except when different colours are being knitted in the same garment (intarsia), although this is still very different from weaving as the threads are not arranged perpendicularly.

[5] The fabric is steamed before cutting, as this prevents it from unravelling

of errors increases. For example, Sirdar Spinning Ltd operates a telephone knitting advice line, and so more complex patterns may impose a customer service burden, with the associated staff costs.

One of the ways that complexity is minimised in knitting is to use a small number of line segments to approximate curves. Thus, accuracy is sacrificed for the benefit of simplicity. In contrast, machines have much less scope for error and infinitely more patience than a human, therefore complex shapes, specified to a higher level of precision, are more likely to be employed in machine-produced garments.

Despite the differences between the processes, there are also commonalities. For example, machine produced knitwear is also made in pieces, and then sewn together ("made up") afterwards [145]. Both designers of hand knitting patterns and commercial garments will work from sketches. However, it is important to realise that although the individual pieces can be represented fairly well in two dimensions, the "made up" garments are 3-dimensional. Fabric puckers and stretches and bands do not necessarily lie flat as stitch patterns have different properties. Therefore, any 2-dimensional representation of a garment is only an approximation.

Chapter 5

Knitwear CAD System

> "Design is a plan for arranging elements in such a way as best to accomplish a particular purpose." - Charles Eames

This chapter explains how the devised software facilitates the knitwear design process. The users will first specify a pattern at a high level, and then refine that pattern at a progressively increasing level of detail. Defaults and heuristics are utilised extensively; these automate common conventions in knitwear design without removing control from the user. Design is a very visual process, and the user is given prompt feedback of the results of their work. As with most other domains, knitwear design has constraints. Both hard and soft constraints will be enforced, the latter being allowed to be relaxed at different levels of granularity. The enforcement of constraints is immediate where practical, in other cases it is acceptable (or indeed necessary) for the design to temporarily be in an invalid state, in which case the enforcement is delayed until appropriate.

First, the requirements for the software are discussed. This is followed by a description of the functionality, and the algorithms that are used to implement this. SEACOP automates the **S**earch, **E**ntry, **A**daptation and **C**hecking **O**f **P**atterns. The entry and checking aspects are described here, whilst adaptation is dealt with elsewhere (see chapter 7). SEACOP also facilitates the persistent storage of designs.

A design is represented on three levels: *questionnaire*, *sketch* and *chart* [97]. The questionnaire and sketch stages are heavily interrelated, and so they are discussed together in section 5.2. Although the chart is produced from the sketch, it is distinct from it. One of the aims of this thesis is to explore the use of case-based design, and since the charts are produced by conventional CAD means alone, they are discussed in appendix B.

5.1 Requirements

SEACOP has been developed without the luxury of a formal requirements specification. The project began with an understanding of the existing design process at Sirdar Spinning Ltd; this is an inefficient process which guided but did not dictate the requirements for SEACOP. The goals of SEACOP are to improve upon this process by automating it, and to facilitate the application of case-based reasoning (CBR) in knitwear design. Although each design is unique, the process is repetitive which suggests some form of automation; however it relies heavily on tacit knowledge which makes it a domain that is suitable for CBR.

5.1.1 Existing process

Visits to Sirdar Spinning Ltd were made in 2008[1]; the objective of these was to understand their design process in detail. This objective was achieved and what follows is a brief description of the process.

5.1.1.1 Process description

Although there is scope for iteration, the Sirdar process is mostly sequential, with the following stages [14, 97, 152]:

1 *Design*. The designer draws on current fashion trends[2] and other inspirations to develop the concept of a new design. The products of this stage are a sketch, and a specification of the design.

2 *Pattern writing*. A pattern writer takes the specification and sketch and constructs the knitting pattern.

3 *Checking*. Manual checking ensures that the pattern is valid and conforms to in-house style rules.

4 *Typesetting*. A typesetter formats the pattern in a format which is attractive for the customer to purchase.

5 *Proof-reading*. A final check is performed to ensure that the typesetting has not introduced any new mistakes.

6 *Sample production*. A garment is knitted in one size, using the pattern.

7 *Sample verification*. Visual checks are performed to ensure the sample conforms to the design.

[1] The underlying processes are expected to stay the same in the years following the visit. 'Out of the box' software to facilitate commercial pattern design to a professional standard are not available. Software exists to design a garment for a knitter, or to design machine-produced garments, but these processes are different from the commercial design of patterns for hand-knitting. Apparently, some of these commercial knitting programs have of the order of 150 person-years of software development [151]. Developing a bespoke software system that was robust enough for their use would be a significant investment for the company.

[2] Although fashions can change radically, everything else about the process remains fairly constant.

The *design* part of the process is the most creative and the designer would use tools such as mood boards and fashion magazines to initiate the process of ideation. They then produce a freehand sketch of the garment, using pencil and paper. This is augmented with a specification of the garment consisting of a single A4 sheet of paper with high-level details of the design such as stitch patterns and yarns used.

The *pattern writing* (2) stage is not termed or viewed as design within the company, but in fact it is a design process, albeit more detailed and mechanistic than the initial design. Initially it is done using pen and paper; typically they use graph paper to construct knitting charts showing complex areas of the garment such as the neck and shoulders. Often, all the sizes are drawn on the same graph paper. Then, they will calculate the numbers of stitches and write this out as textual instructions. These are then typed out using a word-processor.

In the *checking* (3) stage, the processes used to generate the textual instructions will be re-visited and checked to ensure that the written instructions will produce a garment that is consistent with the output of the *design* (1) stage. If there are any mistakes, the pattern is changed and the checks repeated. A pattern is typically checked 2-3 times. Often the checking discovers basic arithmetic errors such as centimetre to inch conversions. The checker is also looking for violations of constraints, such as:

- house style rules must be adhered to
- size, e.g. the width and length must be correct
- shaping stitches must be correctly applied to give the shape in the sketch
- symmetry: most garments are symmetric (or contain many symmetric features)
- buttons must usually be lined up
- the armhole must be approximately the same length as the sleeve
- the pattern must be valid, i.e. the balance between increases and decreases must be correct.

The *typesetting* and *proof-reading* stages are largely concerned with presentation. In contrast, the *sample production* and *verification* processes are the final quality checks performed before the pattern is sold. In reality stitches are not perfectly uniform and fabric sags and stretches so it is difficult to ensure the finished garment has the required appearance without actually knitting it. If the pattern fails verification then alterations are made to the written pattern, if necessary these alterations are checked and then another sample is produced.

5.1.1.2 Discussion

The existing process at Sirdar is largely manual and is very labour-intensive and time consuming. In its entirety, the process typically takes several months and involves many people. Pattern production is therefore expensive (due to the high labour content) and slow; the latter is an issue if the goal is to respond quickly to changes in fashion.

The process involves a mixture of types of task. Some tasks are creative and creativity is traditionally hard to automate or simulate on a computer. Some tasks are more mechanistic, but still involve individual discretion and decision-making, for example fitting a stitch pattern to a shape. Other tasks are trivial, such as centimetre to inch conversions, although even this can be a source of error.

Some parts of the process are well documented, for example there are many books which explain which stitches to use to shape a garment. Other aspects are not well documented and rely on tacit knowledge that is passed on by word of mouth. Sometimes this knowledge is taught to others by example rather than by the exposition of general principles. Such tacit knowledge is notoriously hard to capture in a software engineering process. It also represents a substantial risk for the employer, since when personnel leave the organisation they take their knowledge with them.

It is easy to see how this process can be improved upon and indeed the requirements for SEACOP were that a substantial proportion of the process should be automated.

5.1.2 SEACOP specification

The requirements evolved from an understanding of the existing manual process, as explained below.

The existing process involved much paper documentation and there were three key documents: a specification filled out by the designer (detailing the measurements, yarns, stitch patterns etc.), a sketch of the garment, and knitting charts. This documentation should exist in an electronic form within SEACOP. In the manual process, the specification is a guide used to produce the sketch, and both these documents serve as an input into the pattern-writing process, which produces the knitting charts. Thus, the specification becomes a sketch, and the sketch becomes a chart, and this process flow can be automated in SEACOP, with the major change that instead of the rules for transforming one thing into another being purely manual, they would be automated by the system.

When editing the sketch, it is important that enforcement of constraints is automated; most of these constraints are knitwear-specific, as described in the checking part of the process in section 5.1.1.1. These are soft constraints, i.e. the user must be able to 'turn off' their enforcement, either at a localised level or on a global level. Other constraints were not explicitly stated during interviews with designers but apply to the drawing of any 2-dimensional shape. The user must not have so much freedom that they can turn the sketch into nonsense, for example by leaving a gap in the outline of a shape.

When editing charts, the constraints are clear. A chart must be valid, i.e. the balance of shaping stitches must be correct. The shaping must be correct also, i.e. the shaping stitches at the edge must be consistent with the change in width of a row.

Both sketches and charts must be capable of being produced in a range of sizes. There are established guidelines about the measurements of the sketch, and these must be adhered to. There are two aspects to knitwear grading: scaling up the shape of the sketch into different sizes, and fitting the pattern to the shape. Grading is non-trivial because the scaling used is non-uniform, and many patterns cannot be 'broken'. SEACOP must be able to cope with variations from standard measurements and shapes, and any valid stitch pattern, including ones involving cables. SEACOP needs to be able to make a reasonable attempt at grading any pattern. The results do not have to be perfect, so the user needs to be able to edit both the sketch and the chart.

The mapping from questionnaire to sketch, and sketch to chart do not need to be perfect either, but they do need to conform to established conventions, and be consistent and repeatable (i.e. deterministic). The user should be able to complete a questionnaire and find that the sketch produced matches their expectations, and is in accordance with standard rules for the sizing of garments.

The designers were also clear that they wanted instant (or rapid) visual feedback to their changes. Hence, when the questionnaire is changed in SEACOP, the changes to the sketch must be immediately visible. Both the need for precision and constant feedback must be supported.

5.1.2.1 Limitations

The main limitation of SEACOP is that its output is a knitting chart, rather than textual instructions. The mapping between the chart and textual instructions is not a complex one, but it would require significant effort to achieve natural language generation in the required format, and as this is not expected to lead to major scientific achievement, it is beyond the scope of the work here.

Another limitation is that SEACOP only produces designs for sweaters and cardigans. SEACOP could be modified to allow production of other types of garments with further work. However, the majority (about 86%)[3] of garments featured in knitting patterns are cardigans or sweaters. The main goal of SEACOP is to facilitate case based design; this requires garments to be compared, and as it is only practical to compare garments of the same type, it was decided to focus on the most common type.

Since the goal was to automate the most common patterns, some rarer scenarios such as crochet and circular knitting are not supported.

5.2 Questionnaire and Sketch

This main purpose of this chapter is to discuss the two interrelated stages of pattern design: the *questionnaire* and *sketch*. The next section introduces the main components in the user interface, and how they interact. This is followed by a more detailed treatment of the questionnaire, and the tools used to create design sketches. Finally, the algorithms which transform the questionnaire into the sketch and

[3] When 100 garments were chosen at random from a sample of 1111, 86 of them were found to be cardigans/sweaters.

enforce constraints during sketch editing are explained.

5.2.1 Interactive Sketch Generation

Interactive sketch generation has four main components [14], as shown in figure 5.1:

- *Sketch editor*: This is the main and largest part of the window; it shows the design of the sketch in progress. In the example below it is covered by horizontal and vertical gridlines and shows one of the sleeves, a collar, and the back and front of a sweater. The latter two are obscured by the questionnaire in figure 5.1, although the questionnaire dialogue box can be moved around the screen as required.

- *Side panel*: This is the rectangular area on the left, which allows the user to set various personal preferences, such as whether grid-lines are displayed. Also, it facilitates some presentation/display choices about the content of the sketch, such as buttons.

- *Questionnaire*: This is the dialogue box in the foreground, entitled 'Size' in the example. The questionnaire allows the user to make important choices about the design, such as the dimensions. It is organised in six stages and is discussed in section 5.2.2. The questionnaire is not always visible but when it is, the sketch editor and side panel are inactive as it is a modal dialogue box.

- *Progress bar*: This is the thin rectangular area at the bottom of the window indicating what stage in the questionnaire the user has reached.

As discussed previously, a garment is represented on three levels. As the user moves through the questionnaire, the sketch, and the chart, the level of detail increases. Interviews with knitwear designers showed they prefer systems which are interactive and provide continuous visual feedback. Therefore, the user interface for the questionnaire and sketch stages were integrated; the sketch provides rapid feedback to changes made in the questionnaire.

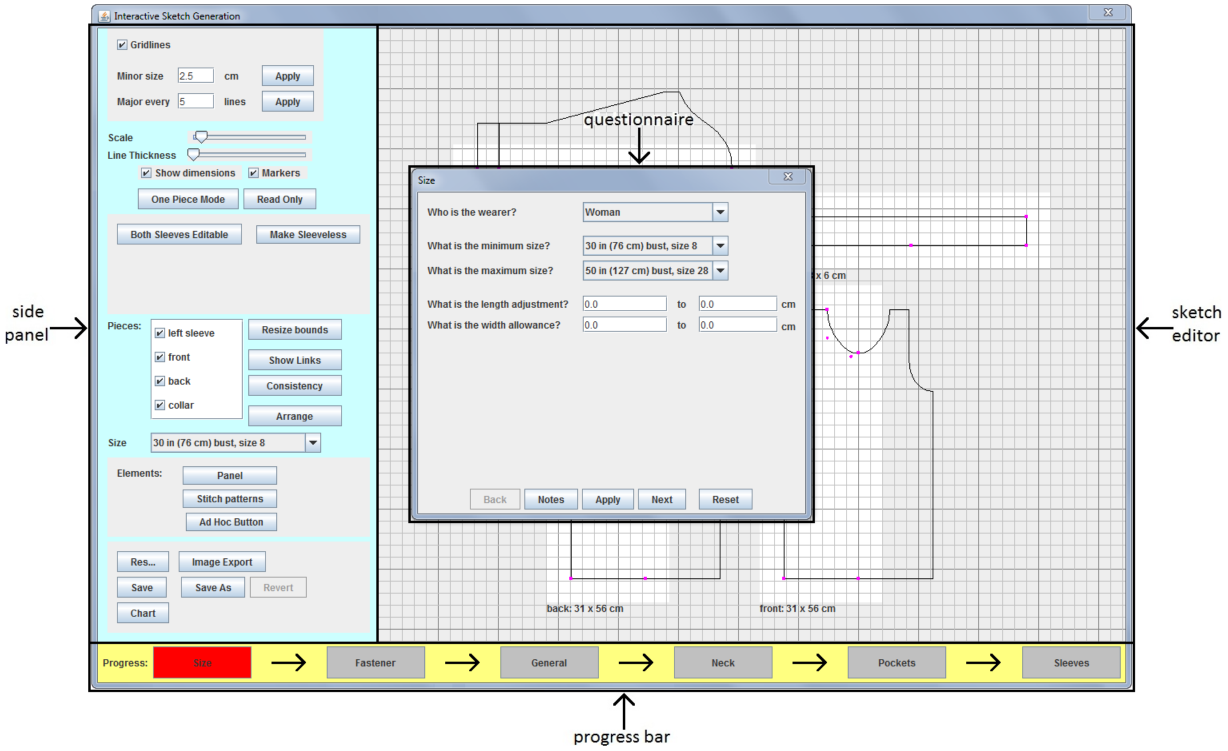
Via the navigation buttons, the user can work through the six stages of the questionnaire, which correspond to those shown in the progress bar:

- *Back* moves back to the previous stage

- *Apply* remains on the current stage, but it causes the changes made (in that stage of the questionnaire) to be applied to the sketch immediately

- *Next* moves forward to the next stage

- *Reset* remains on the current stage, but reverts all the choices in that stage back to the default.



☑ Gridlines

Minor size 2.5 cm

Major every 5 lines

Scale

Line Thickness

☑ Show dimensions ☑ Markers

Pieces: ☑ left sleeve ☑ front ☑ back ☑ collar

Size 30 in (76 cm) bust, size 8

Size

Who is the wearer? Woman

What is the minimum size? 30 in (76 cm) bust, size 8

What is the maximum size? 50 in (127 cm) bust, size 28

What is the length adjustment? 0.0 to 0.0 cm

What is the width allowance? 0.0 to 0.0 cm

x 6 cm

back: 31 x 56 cm

front: 31 x 56 cm

Figure 5.1: Questionnaire and sketch

When the user presses the Back, Next or Apply buttons, the decisions they have made in the questionnaire are applied to the sketch, via the *sketch instantiation algorithm* (as discussed in section 5.2.7.1). For example, changing the gender of the wearer from a woman to a man affects which side the buttons appear on a cardigan. Thus, the sketch provides the visual feedback from the questionnaire choices that designers require; the designer can see the effect of their changes instantly. Some of the questionnaire options have effects that interact with each other, e.g. the location of each button is affected by both the option for the button position, and the option for the number of buttons. This means that providing immediate visual feedback is important, so that the designer is reassured that their choices were appropriate; or alternatively they will be able to see how those choices might need to be changed.

Each button on the progress bar (see section 5.2.2.2) corresponds to a stage in the questionnaire. Once the user has finished the questionnaire, it is no longer visible and they typically progress to the sketch stage, where they use the sketch editor and side panel to make more detailed choices about the garment. For example, they can have fine control over the shape by adding or dragging points on the sketch. The sketch interface is described in more detail in section 5.2.4.

If the user decides that they need to change the decisions that were made at the questionnaire stage, then they can revisit it at any time during the sketch stage by clicking on one of the buttons on the progress bar. This will invoke the appropriate stage in the questionnaire, and the sketch will then be re-instantiated, depending on what choices the user makes. The re-instantiation ensures that the questionnaire and the sketch stages are kept consistent with each other. When the chart editing stage is reached, the questionnaire cannot then be revisited.

Once the user is satisfied that the questionnaire choices were correct, i.e. that the sketch is a good approximation to the design they require, then they are ready to move on to the chart stage. They do this by pressing the Chart button on the side panel. The chart interface is discussed in more detail in appendix B.1.

The CAD system interface aims to satisfy a number of seemingly conflicting requirements. The questionnaire must cause the sketch to be changed immediately, since rapid feedback is required by the designers. However, the sketch must be able to be edited (as described in section 5.2.4), as the output of the questionnaire will rarely be perfect. Also, although the questionnaire is usually a linear process, it must be able to be revisited at any time; the design process is partly iterative and designers sometimes change their mind. The way in which the functionality of the questionnaire and sketch are linked (as described above) supports both a linear and iterative process.

5.2.2 Questionnaire

The questionnaire is the early stage of the design, where the most fundamental decisions are made. The questionnaire is distinct from the sketch, although they are related in the sense that information in the questionnaire is used to construct the sketch. The questionnaire loosely corresponds to the specification stage in the process at Sirdar, explained in section 5.1.1.1.

Whereas the sketch stage involves manipulating shapes (which can be specified at a fine level of detail) using continuous measurements, the questionnaire stage mostly entails choosing from a discrete set of options. Choosing from pre-set options is a comparatively rapid process, compared to manipulating the sketch or chart; so the designer should be able to work through the questionnaire quickly (in a few minutes). However, the number of permutations of options is extremely large (over 10^{15}), meaning the questionnaire facilitates the specification of a wide variety of designs.

Deciding on the composition of the questionnaire presented an important challenge in the development of the CAD system. Including too few options may have meant that the sketch instantiation algorithm (see section 5.2.7.1) had insufficient data to work with. However, including too many options could risk causing user fatigue. The questions which were included were descriptive, using the language of the knitwear designer; and relevant[4] to the sketch instantiation algorithm.

5.2.2.1 Stages of the Questionnaire

There are six stages to the questionnaire: Size, Fastener, General, Neck, Pockets and Sleeves. In the *Size* stage, as depicted in figure 5.1, the user is initially asked to choose the gender of the wearer. This

[4] With few exceptions, e.g. patterns have a reference number which must be recorded this is not relevant to the instantiation of the sketch.

affects the positioning of buttons; on a man's garment they are on the right hand side, but on a ladies' they are on the left. Also, the gender of the wearer affects the range of sizes.

The user specifies the maximum and minimum sizes, and designs will be produced for these and intermediate sizes. The options for sizes are displayed in both metric and imperial measurements, rounded to the nearest integer. The default unit of measurement is always centimetres. However, it is convenient to specify sizes in inches since conventional sizes are even numbers of inches, thus representing an arithmetic sequence. The size of the garment is then used by the sketch instantiation algorithm (as described in section 5.2.7.1) to produce the sketch. Data from a standard table of sizes, taking into account the length adjustment and width allowance, is used to generate the sketch. The latter allow the user to specify looser or tighter fitting garments. For more information about the heuristics used in grading (sizing) knitwear, see section 5.2.7.2.

The next stage of the questionnaire, *Fastener*, is based on the choice of garment (*cardigan* or *sweater*). For sweater there are no further questions, and for cardigan, the type of fastener (e.g. buttons or zip) has to be chosen and then in the case of buttons, their number and position is specified.

The next stage of the questionnaire is *General*. The most significant choice here is that of *background stitch*; see section 5.2.3 for a fuller explanation of the specification of stitch patterns. Also, the user can specify the option for the shape of the waist (e.g. narrow), whether or not there is a yoke and bottom border, and if so what stitch patterns are used for these.

At the *Neck* stage, the user chooses an option for the neck shape; these are explained in section 4.2. They also specify whether the garment has a collar, hood, band or neither. Where a collar, hood or band is specified, the user must choose the stitch pattern that is applicable to it.

The next stage is where the *Pockets* are chosen: their position and stitch pattern will be specified by the user.

The final stage is *Sleeves*. The armhole shape is chosen from a series of standard options; these are explained in section 4.2. Then the user must choose whether the garment is sleeveless or not; if it is sleeveless, then the questionnaire is complete. However, for a garment with sleeves, the user has several choices: sleeve length, whether there are cuffs, and whether the sleeve has straight regions at the top and bottom. The latter determines the default shape of the sleeve (although the user can modify this using the sketch editor).

Finally, the user has to decide whether the sleeves are symmetrical or not. If they are, then only one sleeve is editable, otherwise both sleeves are independently editable. The latter option is rare, since sleeves are almost always the mirror image of one another. It is possible to make the sleeves independently editable at a later stage using the button marked "Both Sleeves Editable" on the side panel; this is shown in figure 5.1.

5.2.2.2 Progress Bar

It is important that the user has completed the questionnaire before moving on to the sketch, since returning to the questionnaire at a later stage results in the sketch being re-instantiated (for reasons discussed in section 7.1.2). The user will be able to visualise their progress through the questionnaire with the progress bar. The progress bar has six buttons, each of which corresponds to a stage in the questionnaire. The background colour of the button indicates the user's progress [14]:

- A *red* button shows that the user is currently at this stage of the questionnaire. For example in figure 5.1 the size stage is active.
- A *light grey* button indicates that the user has not yet visited this stage.
- A *cyan* button indicates that the user has visited this stage, and has changed at least one of the options from the default.
- A *blue* button indicates that the user has visited this stage, but left without changing any of the options from the default value.

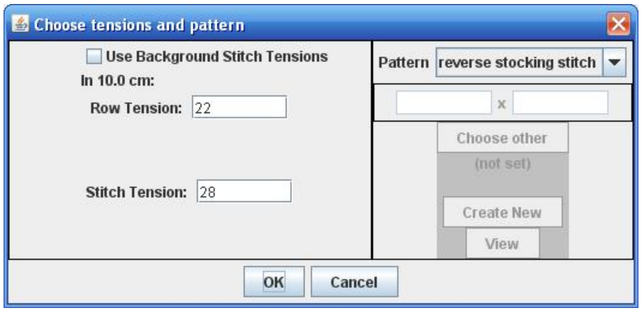
Many usability challenges were addressed when SEACOP was designed. One of these was how to balance the seemingly conflicting objectives of using sensible defaults (to allow the user to make rapid progress) and to avoid errors due to inattentiveness (which could result in designs which do not conform to the specification). Since accepting all of the default values can be an indicator of inattentiveness, the progress bar differentiates between whether a user has changed anything or not.

5.2.3 Stitch Patterns

Depending on the questionnaire options, the user will be asked to choose stitch patterns for specific parts of the garment, e.g. the pockets. At least one pattern must always be specified: the *background stitch*[5]. Stitch patterns are also used sometimes in the sketch editor, for example in the case of panels; this is explained in section 5.2.4.3.

There are several common stitch patterns which have become standard in knitting patterns, e.g. stocking stitch. However, it is very rare for a commercially produced knitting pattern to feature *only* stocking stitch. One of the most difficult aspects of pattern design is fitting a pattern to a shape. So, in order to avoid placing artificial limitations on the designer, both standard and bespoke stitch patterns are available in SEACOP. The following sections explain how to specify stitch patterns in SEACOP; for more information about the use of stitch patterns in knitting, see section 4.5.

[5] The background stitch is usually the most common pattern in the garment, and it is the default pattern, used where no other pattern is specified.



☐ Use Background Stitch Tensions

In 10.0 cm:

Row Tension: 22

Stitch Tension: 28

Pattern: reverse stocking stitch

Figure 5.2: New stitch pattern

5.2.3.1 Choosing Stitch Patterns

SEACOP uses a standard dialogue box which allows users to specify stitch patterns, as shown in figure 5.2. Tensions are an important aspect of knitting patterns, since they control how many stitches are needed to make the required height and length. The left side of the “Choose tensions and pattern” box is dedicated to tensions. If the “Use Background Stitch Tensions” box is ticked, then the tensions for this pattern will be set to the same as the background stitch; otherwise the user must enter a row tension and stitch tension.

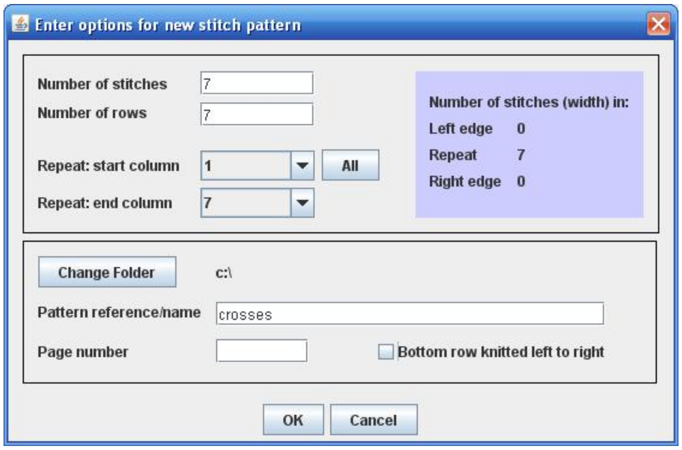
The right hand side is dedicated to setting the pattern; this is done via the drop-down list at the top. In this example, reverse stocking stitch has been chosen, but a variety of other standard stitch patterns are offered. If the user picks the ’rib’ option, then the two boxes below become enabled, they allow the user to set the parameters for an $m \times n$ *rib* (see section 4.5). Alternatively, if the user picks the ’other’ option, then this indicates they require a non-standard stitch pattern. Other patterns, stored on the user’s computer[6], can be accessed by clicking on the ’Choose other’ button. Finally, another option is to click on the ’Create New’ button, which will result in a new stitch pattern being created. More information about the latter is given in the next section.

5.2.3.2 Creating Stitch Patterns

A new stitch pattern can be created by filling the options in a corresponding dialogue box (see figure 5.3) and subsequently editing its knitting chart (see figure 5.4). The size of the pattern will be determined by the number of stitches (width) and number of rows (height). The two labels starting with “Repeat” are used to specify if there is a non-repeating edge; if the user presses the All button then there will be no such edge. In the example shown, the pattern is a repeat of 7 rows and 7 stitches in each row. In the bottommost section, the most important box is the one which requires the designer to enter a unique and descriptive name (“crosses” in this example). The ’bottom row knitted left to right’ check box specifies the direction of knitting (see section 4.5).

The user interface which allows the editing of bespoke stitch patterns is divided into three rectangular

[6] or a network location accessible to it


Enter options for new stitch pattern

Number of stitches 7
Number of rows 7
Repeat: start column 1
Repeat: end column 7

Number of stitches (width) in:
Left edge 0
Repeat 7
Right edge 0

c:\
Pattern reference/name crosses
Page number
□ Bottom row knitted left to right

Figure 5.3: Specifying a new pattern

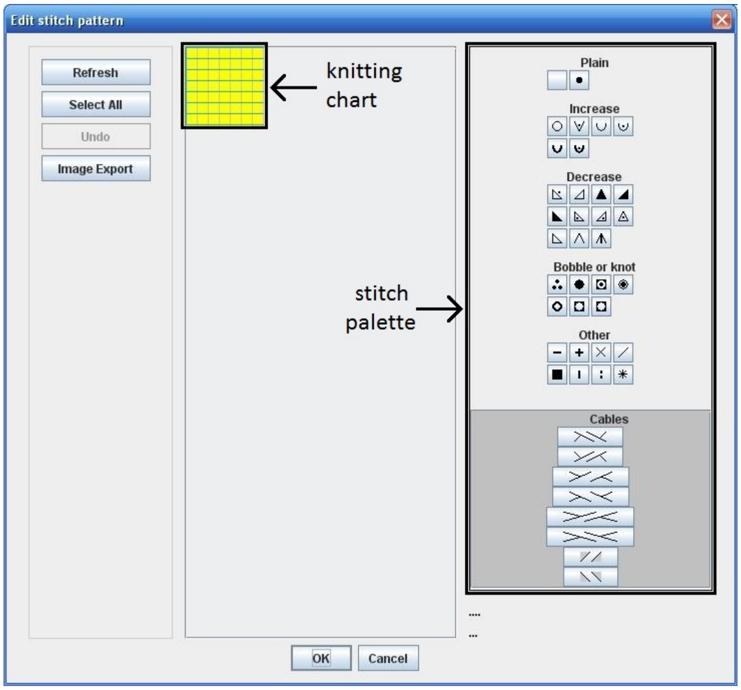


Figure 5.4: Editing a new stitch pattern

regions, as shown in figure 5.4. The leftmost region has buttons which have miscellaneous functionality that improves usability, e.g. Undo. The central region contains the knitting chart for the pattern; each yellow square represents a stitch. The rightmost region is the *stitch palette*, which allows access to all the stitches.

The user can edit the knitting chart by first selecting a region in the chart; this can be either an individual stitch, or a range of stitches. Then, they click on a button in the stitch palette; the stitch(es) that were selected will be changed to those corresponding to the button. The stitch palette is arranged in groups according to the type of stitch, as per feedback from knitwear designers. Designers will typically know which type of stitch they are looking for, e.g. a decrease at the edge, and having them arranged in this way makes it easy to browse through all the stitches of a particular type.

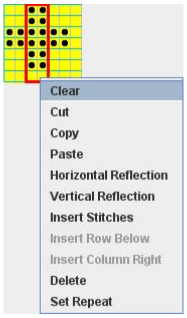


Figure 5.5: Right-click menu on the knitting chart

The procedure for adding cable stitches is slightly different; as these are wider than standard stitches the software ensures that the user can only add them in places that do not violate the integrity of the pattern.

Another means of editing the chart is via the right-click menu, as shown in figure 5.5. Standard operating system clipboard operations are supported, for example: clear, cut, copy and paste. The editor also supports some specialist operations, such as reflection: horizontal reflection uses a vertical axis of symmetry, and vice versa. The user can also employ the right-click menu to alter the size of the pattern. The "Insert Row Below" option is valid whenever an entire row has been selected, and likewise the "Insert Column Right" option is valid when a column is selected. The "Set Repeat" option is valid if whole columns are selected; it sets the repeat period of the pattern, marking any columns not selected as non-repeating edges.

When the designer is satisfied with their pattern they can press OK and it will be created; pressing OK again (see figure 5.3) will set the stitch pattern for that region to the newly created one. The stitch pattern editor is multi-functional and it is comparatively easy to create quite sophisticated stitch patterns.

5.2.4 Sketch Editor

The sketch provides a visual depiction of the shape of the garment which the user can manipulate. As well as the visual objects that make up the shape, the sketch editor contains two kinds of intelligent objects which have specialist functionality: these will be called *move advisers* and *constrainers*. These objects are described in detail in sections 5.2.5.2 and 5.2.5.3, but since they have very profound and far-reaching effects on the interface, a brief description is given below.

Move advisers ensure that the structure of the garment is preserved, e.g. they might restrict a corner to being a right angle. *Constrainers* reinforce constraints on the design, such as restrictions on how a Bézier curve [135] is allowed to be distorted. In some cases, the sketch tool enforces the constraints by disallowing edits which would violate the constraints. In other situations, it is able to "repair" an edit to the sketch so that the constraints are not violated. There are numerous complications; constraints can conflict with each other, they can reinforce each other, and some constraints have a higher priority than others. SEACOP provides a mechanism for turning off some constraints and move advisers (this is presented in section 5.2.4.4).

The algorithms for controlling the sketch are invoked by objects which are added when the sketch is instantiated. The process of instantiating the sketch is discussed in more detail in section 5.2.7.1. However, before these algorithms are discussed, a thorough treatment of the user interface is given below.

5.2.4.1 Moving points

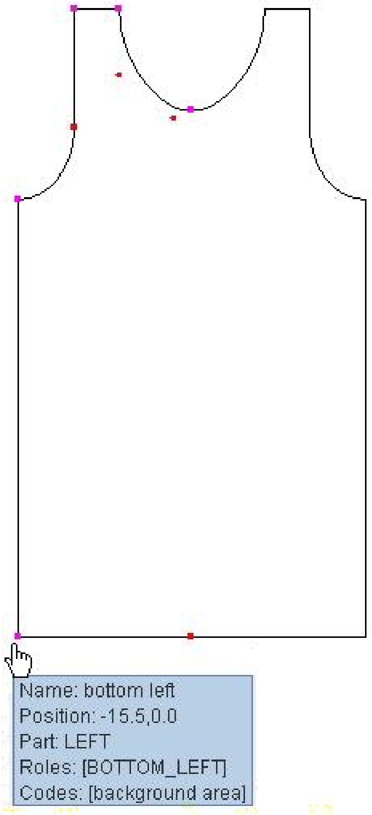
The sketch represents the shape of the pieces in a garment using points, which are joined with lines and curves. The primary means of modifying a sketch is via dragging points. A *marker* denotes points that are able to be dragged by the user. The majority of knitted cardigans and sweaters have a vertical axis of symmetry. Therefore, the sketch is symmetric by default and only points on one side of the sketch can be dragged; the others are positioned automatically by symmetry. Markers are either small magenta squares or red circles, as shown in in figure 5.6. The magenta squares are used to indicate the end of lines and Bézier curves. The red circles are used for the control points of Bézier curves; dragging these alters the character of the curve. Sketches can also contain elliptical curves, these do not require control points and thus only have markers at both ends.

When the mouse cursor is moved so that it is sufficiently close to a point, it changes[7] into a hand symbol, as shown in figure 5.6, to indicate that the point can be dragged. Also, a tooltip appears with information about the point, as shown in the pale blue rectangle.

One of the challenges faced during the design of SEACOP was allowing the user to work accurately and also giving them the flexibility to change shapes by eye. Dragging points works well in the latter scenario, when precision is not essential. However, if the user wants to move a point by an exact distance, SEACOP has an alternative mechanism. The user clicks on the point and it is highlighted; they can then move it by exactly either 1mm (or 1cm if they have also hold the shift key down) by pressing an arrow key. This can be useful, for example, in changing the width of a garment so that the measurement exactly meets some predetermined goal. Also, it is difficult to facilitate non-diagonal movement without this feature.

Often when a point is moved, other points must move with it. For example, the default behaviour in the example shown in figure 5.6 is that the right angle between the point being moved, the one above it (which forms the bottom of the armhole), and the ones to the right of it (which form the bottom of the garment) must be preserved. This behaviour is guaranteed by *move advisers*, which can be turned off (see section 5.2.4.4). Move advisers are explained in section 5.2.5.2.

[7] During normal operation of SEACOP, the default cursor will be in use; this may be operating system dependent but is typically an arrowhead in Microsoft Windows.



Name: bottom left
Position: -15.5,0.0
Part: LEFT
Roles: [BOTTOM_LEFT]
Codes: [background area]

Figure 5.6: Dragging points

Sometimes, a point cannot be dragged to a particular place. For example, the curve for the neck should be a concave region of the sketch, since this will create a hole when the pieces are sewn together. Enforcing constraints (such as this) in SEACOP minimises user error. These constraints are built into the sketch although, as with move advisers, some types of constraints can be turned off.

In addition to the constraints, points cannot be moved outside of their bounding box. The purpose of the bounding boxes is to ensure that points cannot be moved so far that pieces overlap each other. As discussed in section 4.1, a garment typically consists of pieces of knitwear which are knitted separately, then sewn together afterwards. The sketch in figure 5.7 shows three pieces. The size and location of the bounding boxes is not fixed, however. If the user clicks the Resize bounds button, then whilst it remains depressed the bounding boxes are highlighted. During this mode, the user is able to drag the corners of these in order to resize them, as shown in figure 5.7.

5.2.4.2 Adding points

Whilst moving points is the primary means of editing the sketch, it has obvious limitations: unless new points are added, the number of line segments remains static. SEACOP does not restrict designers to only simple shapes, and complex shapes tend to require more line segments (than are created by default) to define them. Allowing the user to draw complex shapes enables the software to cope with changes

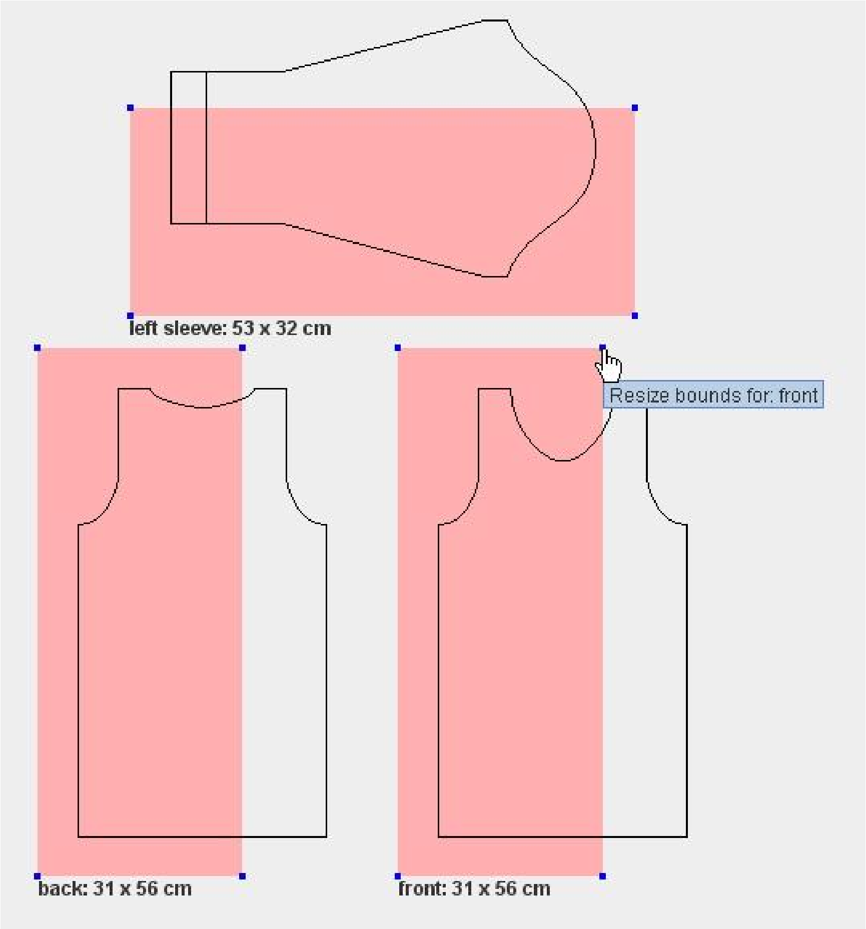


Figure 5.7: Resizing bounding boxes

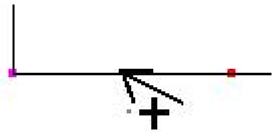


Figure 5.8: Adding a point

in fashion; shapes which are currently unusual may become more common in the future and the users will not be restricted from using these. It also increases the variety of garments that can be produced in SEACOP, providing more scope to demonstrate the application of case based design.

When a new point is added to a line segment, it then becomes two line segments. The user can add a point anywhere[8] on the line, as shown in figure 5.8. Currently, points cannot be added to curves; this limitation could be overcome with further work, since algorithms to subdivide Bézier curves do exist. After a point has been added, it is normally dragged to a new position, in order to modify the shape. The line segments that result from adding a point can, in turn, have points added to them. It would be possible to change a straight line into an effective approximation to a curve, if sufficient points were added.

If a point which has been added[9] (through this procedure) is subsequently not required, then it can be deleted. When the user right-clicks on a point, an option to delete that point is available. Once a point is deleted, the two line segments are replaced with one line segment which joins the points that where connected to it; thus deletion is the reverse of adding a point.

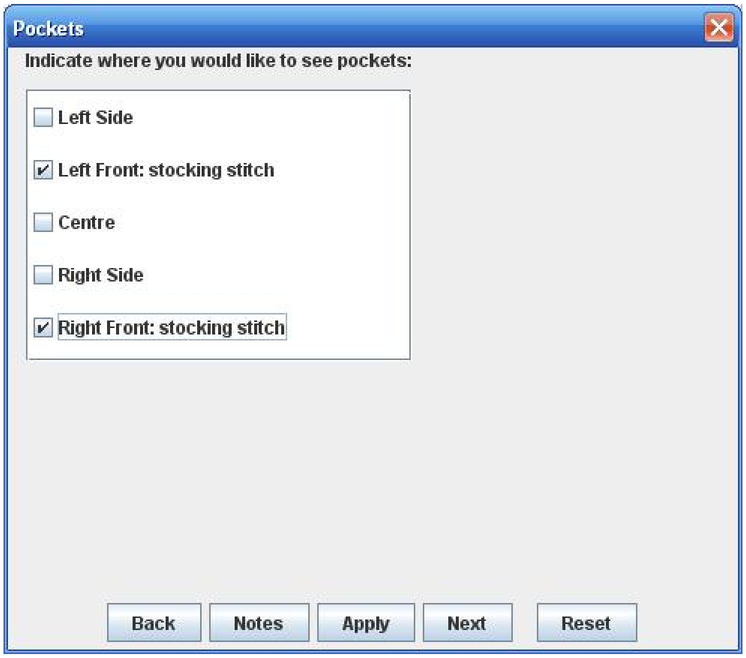
5.2.4.3 Elements

The simplest possible sketch of a piece (of knitwear) will consist of a single polygon. However, more commonly there will be a series of shapes; some will be interconnected, whereas others will be located inside another shape. The boundary between shapes indicates a change of stitch pattern, tension, yarn, direction of knitting, or more than one of these things. Several stitch patterns can coexist in the same piece, allowing the user to create complex garments. The questionnaire contains some options which can result in the creation of extra shapes: these are cuffs (at the end of a sleeve), or borders (on the edge of a piece).

The user can add extra shapes to the sketch through the creation of *elements*. An element is a feature that is added to a piece of knitting; elements tend to be functional or decorative. *Bands* and *panels* are decorative, they are added in order to make the garment appear more fashionable or attractive. *Buttons* and *pockets* are functional elements; buttons act as a fastener and pockets can be used to store items. Of course, buttons and pockets may have a decorative effect also. Elements increase the complexity

[8] As long as the point they want to add is not very close to an end point

[9] Only points which have been added by a user can be deleted. If the user was able to delete the points which were created by default, then the sketch may be changed so much that it becomes inconsistent with the questionnaire.



Indicate where you would like to see pockets:

☐ Left Side

☑ Left Front: stocking stitch

☐ Centre

☐ Right Side

☑ Right Front: stocking stitch

Figure 5.9: Specifying pockets

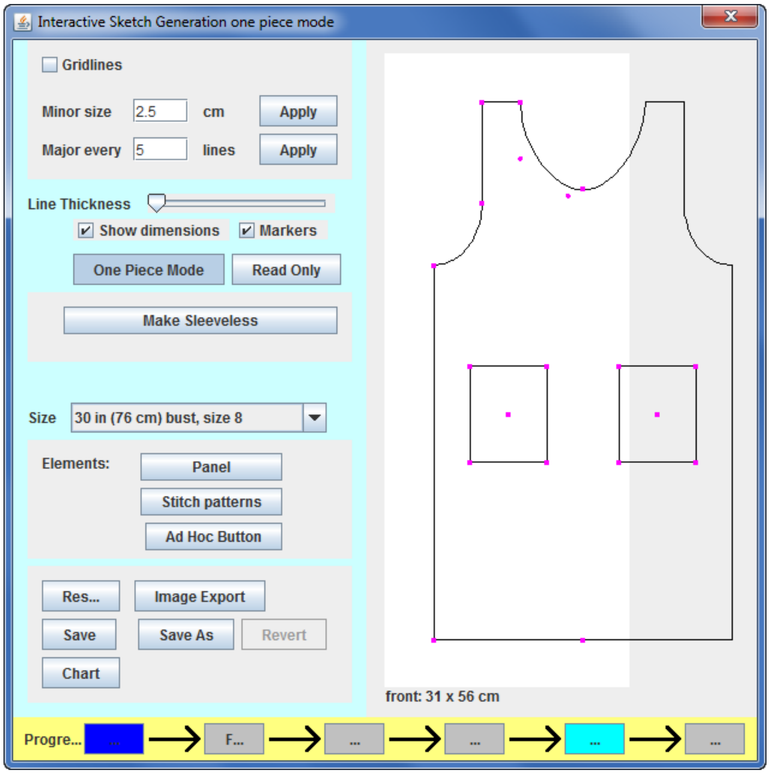
of the garment. Commercially produced knitting patterns tend not to be overly simple, otherwise they would be seen as not challenging enough for the knitter. Allowing this complexity ensures that SEACOP is capable of producing the sort of complex garments that are encountered in the real world.

A *panel* is simply an area which has a different stitch pattern to the area that surrounds it. Panels are initially rectangular, but they can be made into other shapes through the process of adding points (as described in section 5.2.4.2). A panel is used to create motifs or patterns, and can be added to any piece, in any location that the user desires. The user indicates whether the panel is automatically reflected or not; this gives them the flexibility to create asymmetric garments.

Pockets are described in more detail in section 4.6. When the user sets the stitch pattern for a pocket, it is only the top layer which is affected; the designer may wish to change the stitch pattern to achieve a decorative effect. The bottom layer of pockets is always stocking stitch, since this is the simplest stitch to knit, and the bottom layer cannot be seen from the front of the garment.

Another difference between panels and pockets is that pockets have a more restricted location; they can only be added to the front. Pockets are initially positioned according to a heuristic which uses the questionnaire option. For example, figure 5.9 shows the questionnaire stage for pockets; two pockets have been specified, and each one uses the stocking stitch pattern. The heuristic is used to create two pocket elements with a position corresponding to the questionnaire choice, as shown in figure 5.10. Since each pocket can have its own individual stitch pattern[10], it makes sense that each pocket is individually editable. Thus in figure 5.10 both pockets have markers, which indicates that they are editable, despite the fact that the piece has symmetry enforced (the main shape only has markers on one side). However, by

[10] Although in the questionnaire if a second pocket is specified, its stitch pattern will default to that of the first.



☐ Gridlines

Minor size 2.5 cm

Major every 5 lines

Line Thickness

☑ Show dimensions ☑ Markers

Size 30 in (76 cm) bust, size 8

front: 31 x 56 cm

Figure 5.10: Pockets applied to front

default, they are positioned symmetrically through the user of a move adviser. Thus, SEACOP maximises choice by allowing for rare situations (such as a pocket on only one side), but also maximises convenience through the user of intelligent defaults.

There are two types of *button* in SEACOP: standard and ad-hoc. By allowing both ad-hoc and standard buttons, SEACOP provides for automatic enforcement of common constraints (such as buttons being lined up), but also gives the designer freedom to be creative. Like pockets, *standard buttons* are specified in the questionnaire and positioned according to a heuristic. By default they are aligned with each other using a move adviser, as if positioned on an imaginary vertical line. However, the user can disable this restriction (see section 5.2.4.4). *Ad-hoc buttons* are specified outside of the questionnaire, by clicking on a button in the side panel. As their name suggests, the user can place them anywhere, since they usually occur on one side of the front only.

Lastly, the user is able to add *bands* to the garment. A band is represented as a pair of parallel lines, which are either horizontal or vertical. A band could be used, for example, to make horizontal stripes on the front of a sweater. The lines begin and end at the edge of the piece of knitting. Each line has one marker on it, which the user can drag to set the horizontal position if the line is vertical, or the vertical position if the line is horizontal. As with panels, the user can set whether or not a band is reflected, although this option is not applicable if the band is perpendicular to the axis of symmetry.

A comparative summary of the characteristics of the elements is given in table 5.1. There are very

type of element	*added via*	*has own stitch pattern?*	*is reflected?*	*can be added to*
panel	side panel	yes	user-configurable	any piece
band			user-configurable or not applicable	
pocket	questionnaire		via move adviser	front
standard button		no	no	
ad-hoc button	side panel			any piece

Table 5.1: Elements

few restrictions on the adding of elements, so the designer is able to enhance the aesthetic appeal by placing multiple elements to the same garment. SEACOP maximises convenience through automating heuristics (e.g. buttons are evenly spaced), but also places as few limits on the designer's creativity as possible.

5.2.4.4 Relaxing controls and symmetry

As briefly discussed at the start of section 5.2.4, SEACOP contains intelligent objects which enforce constraints (*constrainers*) and preserve the integrity of shapes (*move advisers*); these are discussed in detail in sections 5.2.5.2 and 5.2.5.3 . The objective of this section is to show the flexibility in the sketch editor that results from being able to "turn off" the functionality of move advisers, constrainers, and automatic symmetry. These mechanisms enforce *soft constraints* that are appropriate in the majority of circumstances.

It would be difficult to maintain symmetry by eye in a purely "freehand" sketch editor. If inaccuracies in the sketch were present, it is possible that these could be made more obvious when the sketch was discretised to a chart. This might manifest itself, for example, in having an extra row of knitting on one side of the neck, compared to the other. However, in contrast to *hard constraints*, soft constraints are not universally applicable, and the user needs the reassurance that they are able to deactivate them.

As figure 5.11 shows, when the user right-clicks on a marker they are presented with a standard right-click menu. Of the two options which are applicable, the topmost one (labelled "front") relates to the piece, and the other (labelled "bottom left") is about the point. Each option invokes a sub-menu. The menu for the point contains options that are equivalent to those described below for the piece[11], except the scope of the effects is local to that point. Thus, the designer has a high level of control as to which constrainers and move advisers are applied; they can turn them all off to produce a garment with an unusual shape, or turn only some of them off to produce a more standard garment which has an unusual shape in one specific location.

[11] There is no localised option to remove symmetry, since the points are interconnected it may not make sense to have control over the symmetry of only some of those points.

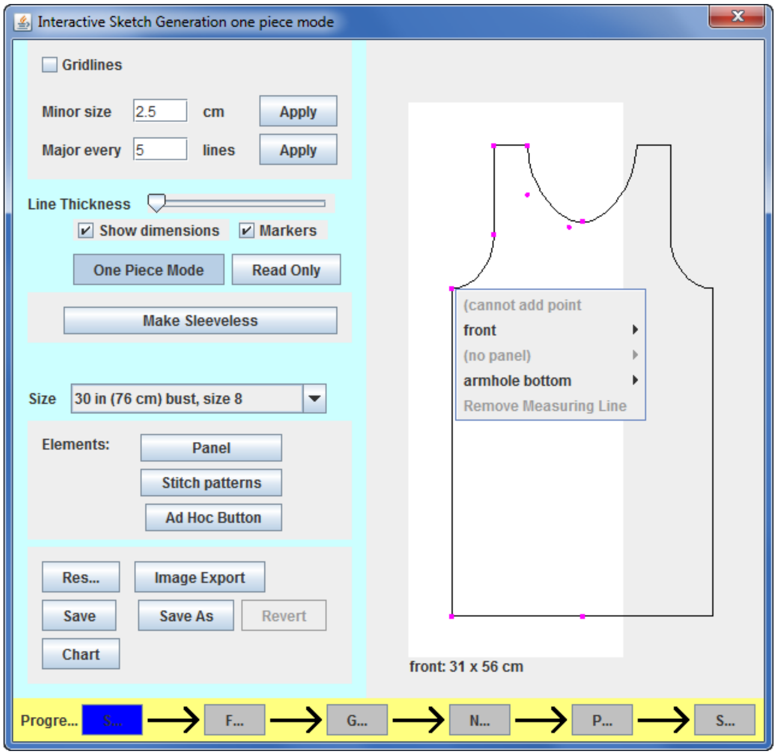


Figure 5.11: Right-clicking on a point

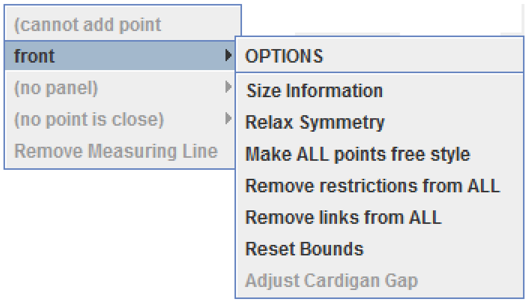


Figure 5.12: Pop-up menu for a piece

The sub-menu for the piece is shown in figure 5.12; it applies changes to the whole piece (in this case, the front):

- The *Size Information* option opens up a new window which is laid out on a grid (like a spreadsheet). It shows the measurements of key regions of the garment, in all the sizes. The designer can use this to check that their garment conforms to established rules about garment sizing (see section 4.3).

- The effect of the *Relax Symmetry* option is to remove the automatic symmetry mechanism, so that there would be markers on both sides of the shape. Initially the shape remains unchanged, but the user is able to drag points on one half of the shape without equivalent points being affected on the other half. This means asymmetric garments can be created, increasing the variety of designs which SEACOP is capable of generating.

- The *Make ALL points free style* option removes most[12] of the the *move advisers* from the piece. This leaves the user free to distort the shape; for example, the angle at corners where two lines join will no longer necessarily be 90 degrees. This allows the designer to create garments which are unconventional or unusual, for example acute angles may be a feature of a particular style. In fact this option can have quite far reaching effects, e.g. buttons might not necessarily be aligned with each other.

- The *Remove restrictions from ALL* option will remove all of the *constrainers* from the piece. This means, for example, that the user will be able to distort the Bézier curves without restriction.

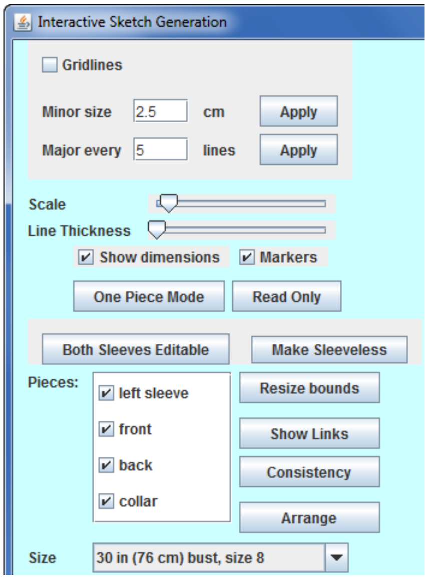
- The *Remove links from ALL* option will remove the *linked points* which involve this piece. This would allow, for example, the back to become larger than the front.

- *Reset Bounds* automatically re-adjusts the bounding box (see section 5.2.4.1) so it accommodates the editable region of the piece, plus a margin.

- *Adjust Cardigan Gap* is an option only available to the front of cardigans; it adjusts the gap between the two halves.

The objects which are deactivated by these menus cannot then be re-activated, except by forcing the whole sketch to become re-instantiated. If the user is given virtually total control over the shapes, they can manipulate the points so that the constrainers are violated in multiple ways, and it may be difficult or impossible to write a general algorithm to restore the move advisers, constrainers and symmetry. It may be tempting for a naive user to relax everything in order to give them maximum flexibility. However,

[12] Linked Points are not effected by this, neither are some high priority move advisers which preserve the basic integrity of the shape (see 5.2.5.1 for an explanation of co-located Shape Points).



□ Gridlines

Minor size 2.5 cm

Major every 5 lines

☑ Show dimensions ☑ Markers

Pieces: ☑ left sleeve ☑ front ☑ back ☑ collar

Size 30 in (76 cm) bust, size 8

Figure 5.13: Topmost part of side panel

with experience and training they will learn that over-zealous approach to turning off controls only leads to an unnecessary increase in their responsibility.

The lack of ability to re-activate the controls or symmetry may not be an impediment, since designers are expected to relax as few controls as possible, and relax them as late as possible in the design process. It is envisaged that designers will use the sketch tool in the following way. Firstly, they use the questionnaire which instantiates the sketch to shapes which are hopefully not too dissimilar to their goal. Then, they will change the sketch until it is as similar to their desired shape, without relaxing any of the controls. After this, they will relax only the minimum number of controls that they need to, in order to achieve their shape. Finally, they will progress to the chart tool.

5.2.4.5 Preferences

SEACOP allows designers to set visual preferences. These support the design process by affecting how the sketch is displayed, rather than its content. The preferences are mostly controlled from the side panel, shown in 5.13.

The *gridlines* settings control the regularly spaced lines which appear behind the sketch in figure 5.1. The user controls whether gridlines are used, what the spacing between them is, and how regularly major gridlines appear. As gridlines have an appearance similar to graph paper which is often used by designers for sketching, they are familiar with their use. Designers can use gridlines to line up objects by eye. Although move advisers line up many objects by default, if these are turned off then the user might require the grid for manual alignment.

The gridlines give a continuous indication of scale. Since they are visible whilst points are being dragged, they provide instant feedback of the measurements which are affected by the changes. If points lie in between the gridlines, then only an approximation of the distance between them is provided. However, SEACOP includes an additional tool (which is accessed by right-clicking on the sketch with the mouse) which can provide the exact measurement between any two points. The scale can be altered by the designer through the use of a slider bar or the mouse wheel button.

During scaling, the invariant point is in the centre of the window. Therefore, in order to focus on one area, translation is also necessary. The user can move the sketch around by using the arrow keys. Pressing the button marked *Arrange* resets the sketch to the default arrangement.

The scale is also affected by *One Piece Mode*. When the user presses the *One Piece Mode* Button, they are asked to choose a piece, and sketch editor 'zooms in' on this piece. This allows the designer to focus on just one piece; for example they might want to change the shape of the sleeve without being distracted by the back and front. Alternatively, there is a list of tick boxes with one option for each piece (in figure 5.13, all options are ticked), and this can be used to show or hide particular pieces.

The *Size* box does not affect scaling but allows the user to specify which size of garment is being viewed. Only the lowest size is editable, as the larger sizes are obtained from it via an algorithm (see section 5.2.7.2).

The *Reset* button (which is not shown in figure 5.13) causes the sketch to be re-instantiated, undoing changes that the user has made using the sketch editor.

5.2.5 Sketch Features

The representation of a garment is illustrated in figure 5.14. A *garment* consists of a set of pieces; each *piece* in turn can contain one or more *elements*. Both elements and pieces consist of a series of *Shape Points*. Section 5.2.5.1 explains the *Shape Point* representation. A shape point can be thought of as an item in a list of instructions of how to draw the sketch. It is designed to simplify the representation of a sketch, since only one type of object needs to be allowed for, rather than having separate objects for points, lines, curves etc.

Following from the definition of a shape point, there is a description of the types of object which control Shape Points; these will be called *sketch controls*. As figure 5.15 shows, there are two types of sketch control: *move advisers* (see section 5.2.5.2) and *constrainers* (section 5.2.5.3). All the features of the sketch (pieces, elements, and sketch controls) are created using the sketch instantiation algorithm that is described in section 5.2.7.1. The sketch controls are an integral part of a piece, in the same way as a Shape Point.[13]

Users are able to 'turn off' sketch controls as was described in section 5.2.4.4. Sketch controls are implemented in a flexible way that could facilitate (with further work) the ability of users to create their

[13] Linked Points (see section 5.2.5.2) affect more than one piece and are therefore a part of the garment as a whole, not a specific piece.

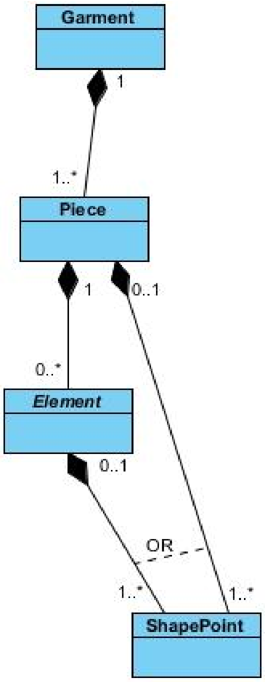


Figure 5.14: Class Diagram of Garment

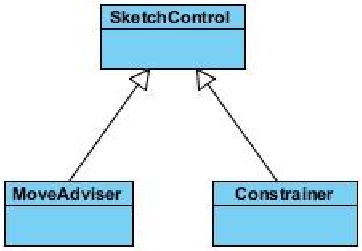


Figure 5.15: Class Diagram of sketch control

own move advisers and constrainers. This would permit the automatic enforcement of constraints which were unexpected at the time SEACOP was created.

5.2.5.1 Shape points

A Shape Point contains two key pieces of information: the *location* of the point and its purpose. Each piece has its own Cartesian coordinate system. When the user hovers over this location with a mouse, the coordinates are displayed. SEACOP allows users access to this information so that they have confidence in the measurements in the sketch.

In addition to the location, a shape point also contains its *purpose*. Shape points are stored within pieces and elements as an ordered list. A shape point and its predecessor in the list act as the specification of part of the shape of the sketch. The most common purpose of a shape point is the end point in a line. However, shape points are seen wherever there are *markers* in the sketch; the example in figure 5.6 shows shape points being used for the control points in a Bézier curve, and the end point of an elliptical quadrant. A Shape Point can have any one of the following as its purpose:

1. The end of a line segment.
2. The end of an elliptical quadrant.
3. A control point for a Bézier curve.
4. A new start point for a line segment, elliptical quadrant or Bézier curve.
5. A button.
6. To specify a horizontal or vertical band.
7. To specify a curved neck band.

If a Shape Point is the end of the line, the start point will be the location of the previous shape point in the list. In categories 1-4, the location of the previous Shape Point is significant.[14] However, if the preceding Shape Point (from one of these categories) would be in an incorrect location, a starting point (category 4) is inserted first instead. Unfortunately, in order to avoid a non-intersect (see section 5.2.6.5) this leads to the presence of *co-located Shape Points* in the sketch; the final Shape Point in a shape is typically in the same location as the first one. Co-located Shape Points are kept in a consistent location through an alignment (see sections 5.2.5.2 and 5.2.6.3).

5.2.5.2 Move advisers

A *move adviser* is a heuristic which causes Shape Points to move in response to changes in the location of other Shape Points. The name "move adviser" arises from a metaphor: the move adviser suggests movement (as a consequence of other movement). Move advisers enforce soft constraints in the design such as corners being right angles. It would be awkward to expect the designer to ensure these were

[14] In categories 6 and 7, only one point is required so specify the line as the other information is calculated by an algorithm.

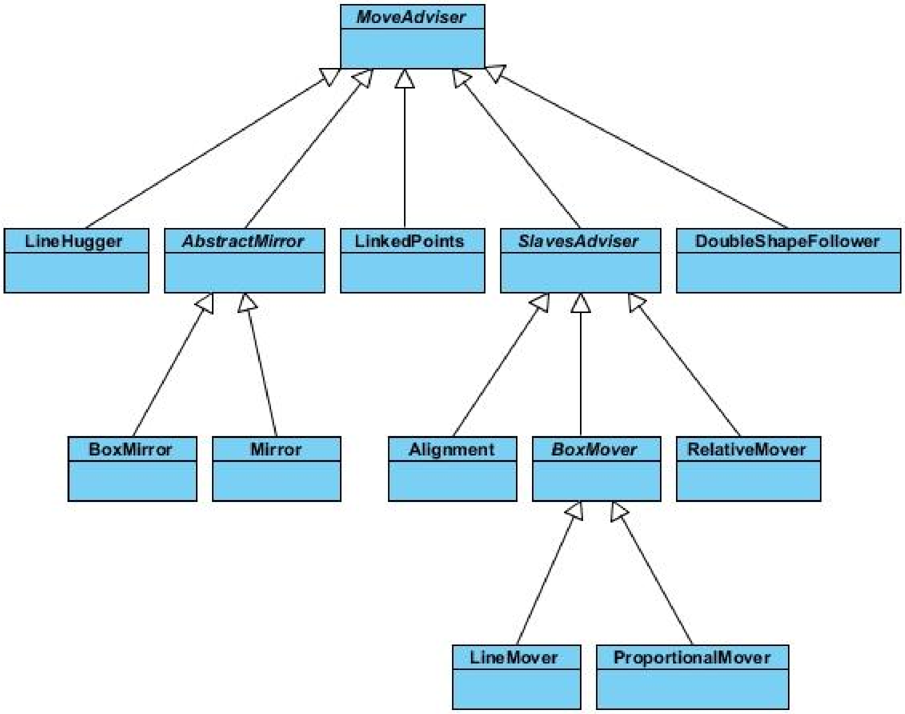


Figure 5.16: Class Diagram: Move Advisers

Type	*Function*
Move Adviser	A type of rule which causes shape points to move in response to the movement of other Shape Points.
Abstract Mirror	Has two reference points and symmetry is relevant but the move adviser is not designed to be reflected.
Slaves Adviser	Has a set (of indeterminate size) of Shape Points that may move.
Box Mover	Keeps one or more shape points in a position relative to a notional rectangle that is defined by two Shape Points.

Table 5.2: Abstracted move advisers

adhered to through manual means. Through the enforcement of these constraints, SEACOP ensures that patterns adhere to common conventions (such as corners being right angles) unless the designer specifies otherwise.

Figure 5.16 shows the hierarchy of different types of move adviser. Each type of move adviser is explained in tables 5.2 and 5.3. A move adviser cannot insist that movement happens, since such movement may violate a constrainer (see section 5.2.5.3). It may also violate a *fundamental constraint*, or move advisers may contradict each other; the resolution of these issues is explained in section 5.2.6.

Figure 5.17 illustrates a proportional mover with a single slave. If either of the two master Shape Points are moved, then the slave Shape Point will be moved so that both $\frac{x_s}{x_m}$ and $\frac{y_s}{y_m}$ are invariant. The exception to this is where $x_s = 0$ or $y_s = 0$, in which case there is no change in the relevant coordinate of the slave. Typically, the master Shape Points are the end points in a Bézier curve, and the slave is a

Type	*Function*	*Example Application*
Alignment	Keeps the horizontal and/or vertical coordinate of a set of Shape Points equal, should any one of the points in that set be moved.	If the bottom corner of the front or back is moved, it moves the other points so it is always a right angle.
Box Mirror	Used where there are two sets of Shape Points which are in positions that are normally reflections of each other, and each set includes one point defined as the origin. If a Shape Point other than the origin is moved, its counterpart in the other set is moved to the same position relative to its origin, taking symmetry into account.	In a pair of pockets, keeps the dimensions the same. When shape points on one side are dragged, a change is made to the corresponding point on the other side.
Double Shape Follower	Two 'parts of a shape', two Shape Points and a direction are specified. When either of the shape points is moved, it ensures that the other is aligned with it in the direction specified, and that each shape point lies on its 'part of a shape'.	When either end of a border is moved, the other is moved so that it remains a straight line and both ends connect with the end of the piece.
Line Hugger	Used where there are two Shape Points which each should be the same distance from their reference shape point along a line which runs parallel to the x or y axis.	Keeps the two lines parallel where there is a neck band for a 'V' neck garment
Line Mover	Used where a set of Shape Points are arranged on a line parallel to an axis, and there are two additional master shape points. If the masters are moved parallel to the axis, or a point in the set is moved perpendicular to the axis, then the relative position of the shape points in the set is preserved.	Keeps the buttons lined up; also keeps them in the same relative position if the height of the front is changed.
Linked Points	Keeps a point in a location which is consistent with a corresponding point in another piece, relative to its surrounding Shape Points.	Keeps the armhole depth consistent between back and front.
Mirror	Where there are two Shape Points that are reflections about an axis of symmetry, moving one shape point updates the position of the other so it remains a reflection.	Each pocket has a local origin point. If one is moved, the other one in the pair is moved to a corresponding position.
Proportional Mover	If two master Shape Points define a notional rectangle, then it keeps a set of shape points which are located inside this rectangle in the same relative position, when either one of the masters is changed.	Preserves the character of Bézier curves (by moving the control points) when they are resized.
Relative Mover	When the master Shape Point is moved, a set of shape points moves with it so they stay in the same position relative to the master.	When the local origin of a pocket is moved, the rest of the Shape Points on that pocket move with it.

Table 5.3: Functions of Move Advisers

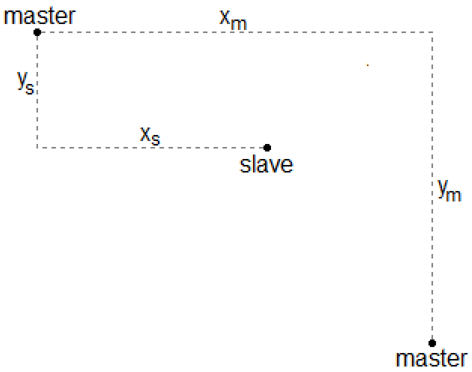
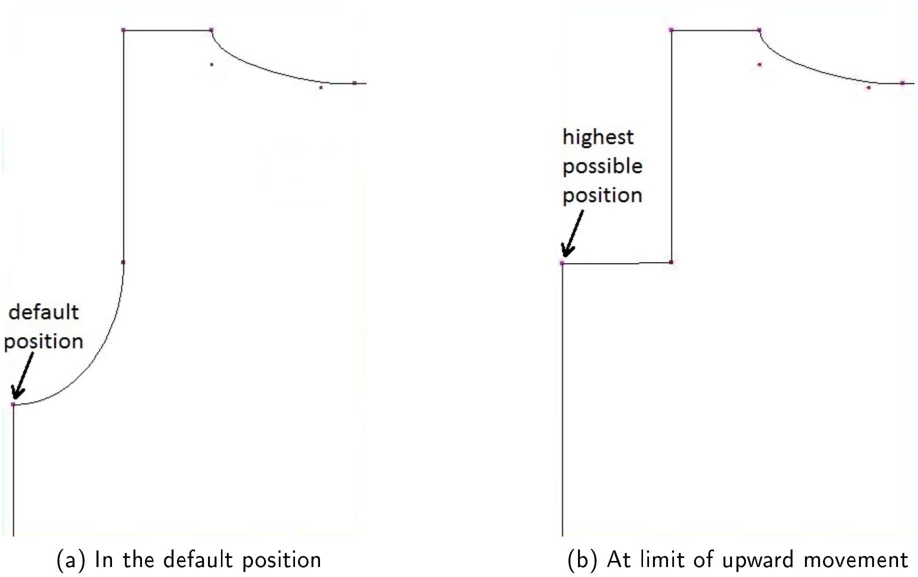


Figure 5.17: Proportional Mover



(a) In the default position

(b) At limit of upward movement

Figure 5.18: Armhole bottom

control point.

5.2.5.3 Constrainers

Constrainers, as their name suggests, forbid particular Shape Points from being moved, according to some in-built logic. Constrainers disallow unconventional edits, which are often the result of user error. There are two types of constrainers, *rules* and *Bézier shape preservers*.

A *rule* prevents a Shape Point from moving in a certain way, relative to another Shape Point. For example, the bottom of the armhole cannot be moved vertically beyond the end of the curve. Figure 5.18a shows the default position of the armhole bottom. In figure 5.18b, the bottom has been moved up as far as it can go. The rule is preventing the shape from being distorted to the point that the shape ceases to be a functioning armhole.

A *Bézier shape preserver* places restrictions on the position of one or both of the two control points that are associated with Bézier curves. A Shape Point that is subject to a Bézier shape preserver will have its location restricted to a rectangle which is defined by the relative positions of the end points of the curve.

A description of move advisers and constrainers has been presented in this section. However, the sketch controls cannot be understood in isolation. As previously discussed, move advisers can 'propose' moves that would violate the constrainers. Conversely, move advisers can propose moves that stop the constraints from being violated; e.g. a proportional mover can move the control points so that a Bézier shape preserver is not triggered. More information about how the features work together is given in the next section.

5.2.6 Managing Movement

Sketch controls and fundamental constraints are part of a coherent system for managing movement in SEACOP. The system ensures common sense and fashion specific constraints are adhered to. However, it also aims to carry out the user's intended action (the dragging of Shape Points) with minimal deviations, in a way which is as consistent and intuitive as possible. All this is automated without the user having to get consciously involved, meaning that they can concentrate on achieving their design goal, and leave most of the details of maintaining integrity to SEACOP.

5.2.6.1 Challenges of constraints and heuristics

Movement is managed in SEACOP through the application of constraints and heuristics. The management of constraints typically poses a number of challenges, for example constraints can have differing priorities. Also, design problems often have both “hard” and “soft” constraints; the latter are not always applicable, and are more like guidelines than constraints.

Constraints can be impractical or impossible to satisfy. They can be mutually exclusive or incompatible, i.e. the *system of constraints* can be impossible to satisfy.

Heuristics are used to ensure constraints are satisfied in SEACOP. When such heuristics are formulated, it is sometimes difficult to envisage their effect in a particular situation. This is particularly true in a system like SEACOP, where a system of constraints is managed by a system of heuristics, whose effects can cancel out or reinforce each other. Applying a heuristic to fix one constraint can cause another to be violated.

The constraints which are described in this remainder of this chapter were formulated following observation of knitwear designers, and study of the outputs of the existing design process at Sirdar. Algorithms for fixing constraints were devised using a pragmatic approach; ideally the fixes will be simple to implement, will make as few changes to the design as possible, and be able to repair any violation.

5.2.6.2 Exploring the consequences

When the user tries to drag a Shape Point, this is known as a *proposed move*. *Proposed moves* can also be instigated by move advisers, rather than the user directly. The reason for the term *proposed* move is that they may be disallowed, if they violate any of the restrictions in place (e.g. bounding boxes). The object of the *exploring the consequences* algorithm is to build the *list of proposed moves (LPM)*. The first entry in the *LPM* will be the user's attempt to drag the Shape Point. Subsequent entries, if any, will be supplied by the move advisers. Any of the *proposed moves* can violate restrictions, and depending on the nature of those violations this may result in the whole movement operation being disallowed. Therefore, no changes to the sketch are made until the *LPM* as a whole is checked, as an atomic operation.

A move adviser is said to *fire* if it leads to one or more proposed moves. The specification of a proposed move consists of the shape point to be moved, and the x and y coordinates of the proposed location. Some move advisers are designed to only supply one of the x or y coordinates; in this case the missing coordinate is unaffected unless it is supplied by another move adviser. For example, an Alignment may line points up vertically but their horizontal position could be unaffected.

As well as the LPM, the exploring the consequences algorithm takes the *set of proposed move advisers (SAMA)* as its input. Initially, this just consists of all the move advisers for the garment, but move advisers can be removed from SAMA without actually being removed from the garment (as described below).

Exploring the consequences is detailed as algorithm 5.1. It begins with an LPM with one proposed move, then iterates through the SAMA, which is prioritised according to the algorithm described in section 5.2.6.3. The move advisers are listed in descending order of priority so the most important are examined first. If a move adviser fires then this means the priorities have to be reassessed, since the prioritisation algorithm is dependent on the state of the LPM. So, a move adviser firing causes another iteration of the algorithm to begin with prioritisation again. Thus, an initial proposed move by the user can trigger further proposed moves, which can in turn cause more proposed moves to be added the LPM, and so on. Exploring the consequences terminates when all move advisers are examined and none have fired.

Exploring the consequences is of quadratic complexity. If SAMA is of size n, then assuming move advisers only fire once, and it is the lowest priority adviser which fires each time, then a move examiner will be examined $\frac{n}{2}(n+1)$ times. However, in a typical scenario the high priority move advisers fire; and as there are only a small number of move advisers (25 for the default garment), usually only a small number of iterations are required before the algorithm terminates.

Note that a move adviser can only fire once since when it fires it is removed from SAMA; this prevents the algorithm from being stuck in an infinite loop. Since the firing of a move adviser causes the priorities to reset, then in order for exploring the consequences to terminate, it must have examined all the move advisers in SAMA without any of them firing. This ensures that the LPM which is output is complete,

Algorithm 5.1 Exploring the consequences

```
procedure exploreConsequences
inputs: SAMA, LPM

i := 0
while i < size of SAMA
  prioritise(SAMA)
  fired := false
  while not fired and i < size of SAMA
    if SAMA[i] fires then
      fired := true
      add the moves from SAMA[i] to the LPM
      remove move adviser i from SAMA
      i := 0
    else
      i := i+1
    end if
  end while
end while
```

as no further moves were added during the final iteration of the algorithm. The LPM then needs to be checked against the constraints before any actual changes to the sketch are made.

5.2.6.3 Prioritisation

The prioritisation algorithm compares move advisers by considering three important values: level, relevance and extent. The *level* is designed to reflect the inherent importance of a move adviser:

- *Level two* move advisers are those alignments which are used to fix two Shape Points as being in the same location. See section 5.2.5.1 for a description of *co-located points*.
- *Level one* move advisers affect more than one piece; in practice this means *linked points*.
- *Level zero* refers to all other move advisers.

The *relevance* is a value in the range [0,1], which is designed to reflect the extent to which the move adviser is influenced by shape points which are in the LPM. The *relevance* approximately correlates with the proportion of the points that the move adviser is dependent on that are found in the LPM. Thus, if all of the points that a move adviser are dependent on are in the LPM, the relevance will typically be 1, and if none are it will typically be zero. The *relevance* is a heuristic; it is not a simple matter of determining which of the shape points that trigger a move adviser are included in the LPM, because multiple changes can cancel out or reinforce each other. It is not feasible to factor in all these combinations of interactions as the calculations need to be efficient.

The *extent* of a move adviser is a count of the number of shape points that are potentially affected by it. Unlike *relevance*, *extent* does not take into account the current state of the LPM.

Algorithm 5.2 Prioritisation of move advisers

```
function: prioritise-move-advisers
inputs: move adviser m1, move adviser m2, LPM
output: move adviser which has the highest priority

if level(m1)=level(m2) then
  if relevance(m1, LPM)=relevance(m2, LPM) then
    if extent(m1)=extent(m2) then
      (it is unlikely that level, relevance and extent are all equal)
      output an arbitrary choice between m1 and m2
    else
      output whichever of {m1,m2} has the greatest extent
    end if
  else
    output whichever of {m1,m2} has the greatest relevance
  end if
else
  output whichever of {m1,m2} has the greatest level
end if
```

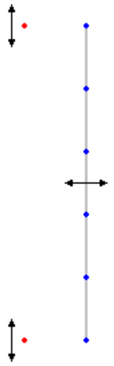


Figure 5.19: Line Mover change level

Exploring the consequences considers move advisers in descending order of priority, since later moves cannot overwrite earlier ones.[15] In order to sort SAMA, it is necessary to determine which of any two move advisers have the highest priority; an algorithm 5.2 accomplishes this. Move advisers are prioritised by comparing their level, then their relevance, then their extent, as necessary.

The level is designed to reflect the inherent importance of some move advisers; level two move advisers cannot be turned off by the user and are therefore the most important. Level one move advisers can affect more than one piece so deserve precedence over level zero advisers, which just affect a single piece.

The relevance is important as move advisers will only 'fire' once (per invocation of exploring the consequences), so it is important that the effect of a move adviser is delayed until the maximum applicable information has been obtained. Move advisers which are less relevant (because fewer points that they are dependent on are in the LPM) will be invoked later on in the execution of the algorithm. However, crucially, it is possible that their relevance will increase as more shape points are added to the LPM.

[15] Except if only one coordinate has been supplied, then a later *firing* can supply the other.

Since each type of move adviser works differently, the *relevance* is necessarily calculated in different ways. An example is shown in figure 5.19 for a Line Mover. A general description of how a Line Mover operates was given in section 5.2.5.2. The implementation of LineMover is designed to be generic in the sense that it can be used to keep any set of points arranged in a line. However, this functionality is only currently utilised in the context of buttons, as described below.

On the left there are two reference points; the topmost of these is known as the margin point. On the right, located on a notional vertical line segment, is a row of buttons; there is one shape point per button. There are three situations which trigger movement, as listed below. For a Line Mover, a contribution of $\frac{1}{3}$ to the *relevance* is made for whichever of the below conditions are true.

1 If the margin point is moved horizontally then line segment (i.e. row of buttons) is moved so that the horizontal distance between it and the reference point remains invariant.

2 If either of the reference points is moved vertically then the line segment is moved vertically, i.e. the buttons move closer together or farther apart on the same line. Note that the topmost and bottommost buttons do not necessarily have to be aligned vertically with the reference point. The buttons are moved so that the position of the topmost and bottom and end points remains the same, relative to the reference points. The buttons do not have to be equally spaced but their spacing relative to the topmost and bottommost button is kept the same during vertical moves of the reference point.

3 If any of the buttons are moved horizontally then the line segment is moved horizontally so that the other buttons follow it.

Unlike *relevance*, *extent* does not take into account the current state of the LPM. The move adviser with the greatest *extent* has the highest priority, since they typically result in more potential moves added to the LPM. This can mean that the relevance of more then one move adviser in SAMA increases; the earlier that this happens the better, since the algorithm can then make better decisions about prioritisation of those move advisers.

In the example of an alignment, the *extent* is is $s - 1$, where s is the size of the set of shape points that it controls. The reason for this is that when one shape point in alignment moves, the others in the set follow. The *relevance* of an alignment is one if any of its points have moved perpendicular to the axis of alignment, and zero otherwise. Thus, alignments tend to *fire* early as they often have both a high *extent* and *relevance*.

The overall effect of the prioritisation scheme is that linked points tend to be considered first, as they are level one; so movement spreads out from the front to the back or vice versa. Then the alignments tend to *fire*, followed finally by the more complex move advisers such as proportional movers and line movers.

Algorithm 5.3 Testing a LPM

```
function: testLPM
input: LPM
output: string indicating success or failure

if LPM violates any constrainer then
  output "unfixable fail"
else
  if LPM violates any bounding box then
    output "unfixable fail"
  else
    if LPM violates any fundamental constraint then
      output "violates fundamental"
    else
      output "success"
    end if
  end if
end if
```

The actual sorting of SAMA is implemented using a Java API [153] method which uses a modified merge sort. Merge sorts have worst case execution time of $n \log n$ [154], where n is the size of SAMA.

5.2.6.4 Movement

SEACOP takes a LPM (which was output from exploring the consequences) as its input and checks it against the restrictions that are in effect, as indicated in algorithm 5.3. If any of these checks results in failure, then the remaining checks are skipped. The result of the test is used in the movement algorithm, as explained below.

The key movement process in SEACOP is shown as algorithm 5.4. If the result of testing the algorithm was success, then the user will see the markers move in response to their actions. However, if the test resulted in a failure, then SEACOP may attempt to remedy the situation by finding a non-diagonal alternative (see below) or a fix (as described in section 5.2.6.5). If these are unsuccessful then the move will *fail*, and the user will see no change. Failure could mean that SEACOP has successfully prevented an error. However, if the move was intentional then the user can override the bounding boxes or constraints (as explained in sections 5.2.4.1 and 5.2.4.4), and attempt the move again.

If a proposed move is unsuccessful and diagonal, then SEACOP attempts to find a *non-diagonal alternative (NDA)*, where the user's proposed move is substituted for one which is parallel to the horizontal or vertical axis. If the proposed move is shown as a vector, then the two *NDAs* will each use only one of the components of that vector. Figure 5.20 shows a simple example of a pentagon which exists inside a bounding box, shaded in grey. There is an attempt to drag a shape point to a new location, labelled *a* and this move fails since it violates the bounding box; however the non-diagonal alternatives *b* and *c* are considered. Since *b* is inside the bounding box, it is chosen instead.

Algorithm 5.4 Movement

```
procedure: movement
input: LPM

initialOutcome := testLPM(LPM)
if initialOutcome = "success" then
  apply the moves in LPM to the sketch
else
  if LPM has diagonal initial move then
    success := nonDiagonalAlternative(LPM)
  else
    success := false
  end if
  if not success and initialOutcome="violates fundamental" then
    fundamentalFix(LPM)
  end if
end if
```

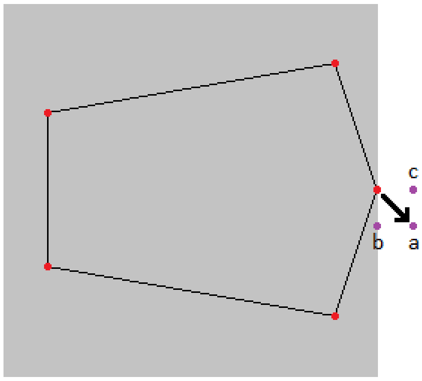


Figure 5.20: Finding a diagonal alternative

Algorithm 5.5 Non-diagonal alternative

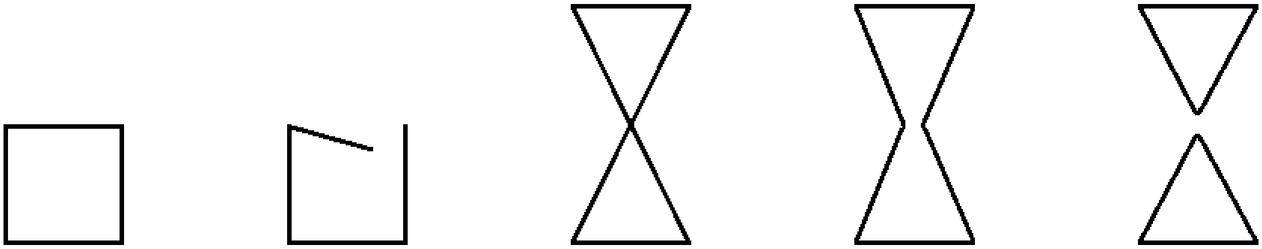
```
function: nonDiagonalAlternative
input: LPM
output: true if a non-diagonal alternative was found and applied,
    false otherwise

(lpmX and lpmY initially consist of just one move)
lpmX := LPM with just the horizontal component
lpmY := LPM with just the vertical component

exploreConsequences(lpmX)
exploreConsequences(lpmY)

if number of moves in lpmX = number of moves in lpmY then
  if distance(lpmX) = distance(lpmY) then
    firstLPM := arbitrary choice of {lpmX, lpmY}
  else
    firstLPM := whichever of {lpmX, lpmY} has the lowest distance
  end if
else
  firstLPM := whichever of {lpmX, lpmY} has the lowest number of moves
end if

secondLPM := whichever of {lpmX, lpmY} was not assigned to firstLPM

if testLPM(firstLPM) = "success" then
  apply the moves in firstLPM to the sketch
  output true
else
  if testLPM(secondLPM) = "success" then
    apply the moves in secondLPM to the sketch
    output true
  else
    output false
  end if
end if
```

Algorithm 5.5 shows the process of attempting to find a successful *non-diagonal alternative*. When exploring the consequences is invoked for the firstLPM and secondLPM, these are equivalent but entirely separate process (starting with a new *SAMA*) to when it was invoked for the original proposed move. It is possible for the *NDAs* to fail for reasons apart from the bounding boxes; for example they might violate a constrainer or one of the moves that is a consequence of a move adviser could do so.

5.2.6.5 Fixing violations of fundamental constraints

Once exploring the consequences has produced an LPM, this must be checked against constraints to ensure that all the moves are permissible. Moves may be disallowed because they violate constrainers (see section 5.2.5.3), or a *fundamental constraint*. Fundamental constraints consist of two types of configuration: *crossovers* and *non-intersects*.

The aim of the repair process is to modify a LPM so that violations of the fundamental constraints



(a) Allowed

(b) Non-intersect

(c) Crossover

(d) Allowed

(e) Allowed

Figure 5.21: Example shapes

are removed. The preliminary to this process is the compilation of the list of *shapes*. Each line, Bézier curve or elliptical quadrant from the sketch, forms a *shape*, and it will have a shape point at either end. A *shape* may be involved in a crossover or non-intersect.

Figures in the sketch represent the outline of the outside of the piece, or a boundary between one stitch pattern and another[16]. Figure 5.21a illustrates a simple rectangle, which is an allowed configuration; it might represent a pocket, for example. Non intersects, as shown in figure 5.21b, are disallowed since they leave it unclear as to what the outline or boundary is.

By definition, the boundary of the outside of a shape can never cross over itself, so a crossover in the boundary should not be allowed. Neither is there a need to allow crossovers in shapes inside of the piece, since the same effect can be achieved without crossovers. The kite shape shown in figure 5.21c has a thickness of zero[17] at the point of crossover, and this cannot be realised in stitches. It would be implemented as either two separate shapes or one shape with a narrow 'bridge' area, as per figures 5.21d and 5.21e.

In addition to constrainers and fundamental constraints, bounding boxes are also a constraint on movement. Bounding boxes were explained in section 5.2.4.1. The differences between the three concepts are summarised in table 5.4. The proposed moves in the LPM will be subject to all three types of restriction.

Algorithm 5.6 shows the process for fixing violations of fundamental constraints. The first stage is *crossover repair*. The algorithm repeatedly finds the highest priority shape which has a crossover; if this does not exist then it moves on to *non-intersect repair*, which again utilises a prioritisation scheme. Either repair process may encounter a violation which cannot be fixed, in which case the process terminates with no moves being applied. Otherwise, the algorithm terminates with the moves in the repaired LPM being applied to the sketch.

Each time a violation is repaired, exploring the consequences is invoked. In reality, fundamental violations tend to only occur when the move advisers and constrainers have been turned off. Therefore,

[16] Or a change in tension, colour or yarn.

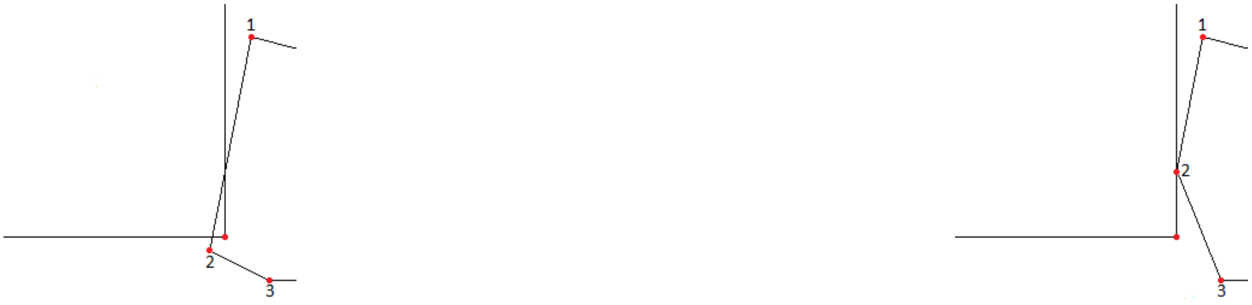
[17] Since the lines in the sketch are assumed to be of zero thickness.

Aspect	*Constrainer*	*Fundamental Constraint*	*Bounding Box*
Scope	As defined; in practice, local (although linked points affect two pieces)	Global	A piece, e.g. the front
Specification and creation	Specified when created, during the sketch instantiation algorithm	Specified as general principles; see explanations of non-intersect and crossover	Created during sketch instantiation algorithm, using a default margin
Can be turned off?	Yes, by the user	No	No, but the user can edit the box

Table 5.4: Differences between a *constrainer, fundamental constraint* and bounding box

Algorithm 5.6 Fixing violations of fundamental constraints

```
procedure: fundamentalFix
input: LPM, list of shapes

while there are crossovers
  priorityCrossover := highest priority crossover in the list of shapes
  if priorityCrossover is a curve-curve crossover then
    terminate the procedure with no moves being applied
  else
    if the line is crossed over once then
      move whichever of the end points is closest to the crossover point
    else
      find largest line segment on the line bounded by crossover points
      shorten the line to become identical to the line segment
      exploreConsequences(LPM)
    end if
  end if
end while

while there are non-intersects
  priorityNonIntersect := highest priority non-intersect in list of shapes
  if priorityNonIntersect is unfixable then
    terminate the procedure with no moves being applied
  else
    stretch the line in both directions until both ends intersect
    exploreConsequences(LPM)
  end if
end while

apply the moves in LPM
```



(a) Before

(b) After

Figure 5.22: Crossover repair

the move advisers that *fire* (if any) tend to be level two, which are used to manage co-located points.

A crossover is fixed by shortening the *shape* (line):

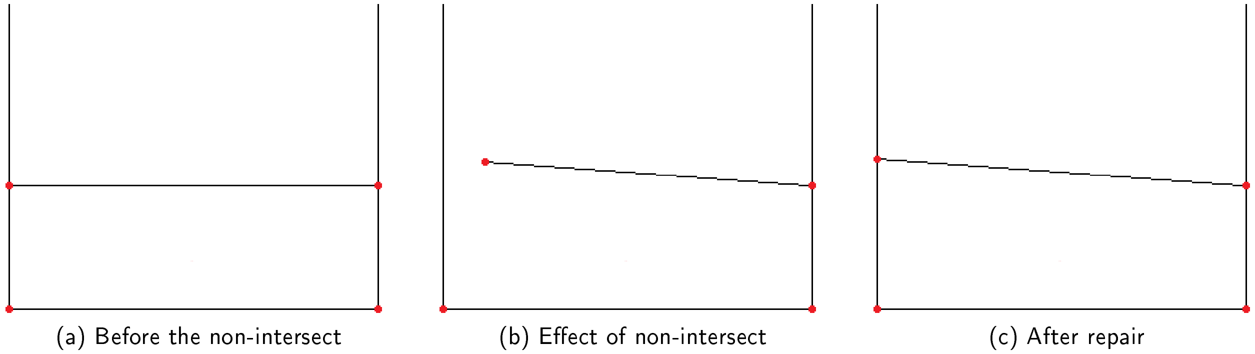
- If the line is crossed over once, then the fix is to move one of the shape points (at the end) to the crossover point; the one which is closest is chosen.
- If the line is crossed over two or more times, then the fix is to find the largest line segment which is bounded by crossover points, and to move the end shape points to the ends of this line segment.

Figure 5.22a shows an example of two intersecting shapes (only the relevant part of each shape is shown). The resulting repair is shown in figure 5.22b; the shape point labelled 2 has been moved back towards 1, to avoid the crossover. Note that fixing one crossover can eliminate others; figure 5.22a features 3 *shapes* with crossovers.

As with crossovers, non-intersects in SEACOP tend to occur where the move advisers and constrainers have been turned off. More specifically, a common cause of a non-intersect is where there is a border, and the end of the border has been moved away from the outline of the piece. Figure 5.23a shows such a typical scenario: a bottom border. In figure 5.23b, the effect of the non-intersect can be seen: note that the user will never actually see a sketch like this since it is invalid. The process of repairing non-intersects involves moving the shape point (which has become disconnected) so the line becomes longer and intersects with something. Figure 5.23c shows the effect of the repair.

It is possible for the 'repair' process for a non-intersect or crossover to simply revert all or part the sketch back to the situation before the move was proposed. For example, an attempt may be made to change the shapes in figure 5.23a to those in 5.24. The 'repair' effectively disallows the move.

Algorithm 5.7 shows how the highest priority crossover is found. Only lines are changed, curves remain static unless they happen to be moved by a move adviser. Curve-curve crossovers are (currently) unfixable so the output does not matter; line-curve crossovers take precedence over line-line since the latter normally has more options for fixing them (either line can be moved). Repairs involving less movement of the



(a) Before the non-intersect (b) Effect of non-intersect (c) After repair

Figure 5.23: Non-intersect repair

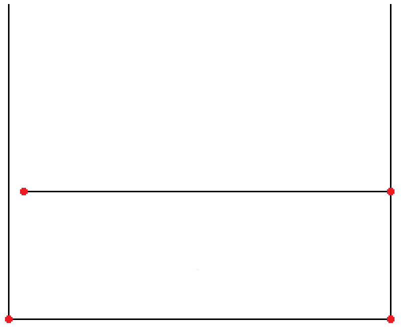


Figure 5.24: Fix which will revert

end points take precedence since part of the goal is to have minimal deviation from the user's intended actions.

Algorithm 5.8 for prioritising non-intersects is similar; as with crossovers, only lines are altered. A non-intersect is unfixable if the line that its line segment lies on intersects with no other line.

It would be possible, with further work, to extend the repair algorithms to include the ability to 'fix' curves, and perhaps to find more intuitive solutions. For example, in figure 5.22, a better 'fix' may be to move point 2 a little towards 3. However, it is difficult to envisage every possible geometric configuration. Most moves the user makes will be valid; a small proportion of these will be disallowed by the constrainers. If the user turns off the constrainers, then these repair algorithms are invoked as a last resort, so that the integrity of the sketch is preserved.

Although it is envisaged that the repair algorithms are only rarely invoked, they are important. If there were no non-intersect fixing algorithm, in a situation when the move advisers are turned off and the sketch is as shown in figure 5.23a, it would be impossible to alter the width of the garment. This is because only one of the two points on the left could be moved at a time, and the scenario shown in figure 5.24 could never be allowed, since it would cause an ambiguity as to where the bottom border ends and the background stitch begins. Such ambiguities could be problematic when the knitting chart is generated.

Algorithm 5.7 prioritising crossovers

```
function: prioritise-crossover
inputs: crossover x1, crossover x2
output: crossover which has the highest priority

if both crossovers have the same configuration then
  if the configuration is curve-curve then
    output an arbitrary choice
  else
    (d1 and d2 measure the change in the line when it is fixed)
    d1 := combined distance which end points of x1 move on fix
    d2 := combined distance which end points of x2 move on fix
    if d1<d2 then
      output x1
    else
      output x2
    end if
  end if
else
  if one of {x1,x2} is line-curve then
    output whichever of {x1,x2} is line-curve
  else
    output whichever of {x1,x2} is line-line
  end if
end if
```

Algorithm 5.8 prioritising non-intersects

```
function: prioritise-non-intersect
inputs: non-intersect n1, non-intersect n2
output: non-intersect which has the highest priority

if both non-intersects are curves OR both non-intersects are unfixable then
  output an arbitrary choice
else
  if one of them is a line then
    output the line
  else
    (d1 and d2 measure the change in the line when it is fixed)
    d1 := combined distance which end points of n1 move on fix
    d2 := combined distance which end points of n2 move on fix
    if d1<d2 then
      output n1
    else
      output n2
    end if
  end if
end if
```

5.2.7 Sketch Issues

In the existing manual process at Sirdar (see section 5.1.1.1), a written specification of the design is produced. The specification in SEACOP is dual-purpose: it provides a valuable mechanism for case-based reasoning, and it is also the means by which the sketch is generated. Designers are free to modify SEACOP's sketches in order to produce what is in their mind's eye, in response to fashion trends and market demand. However, the sketch generation process, as explained below is carefully designed to produce a sketch that is as close as possible to their end goal, to minimise the editing that the user will have to perform manually.

This section also explains the heuristics which automate the re-sizing of the garment (sketches are typically produced in a range of sizes). When sketches are generated, by default they are symmetrical and both sleeves are identical, however this can be changed if required. Finally, SEACOP's method of ensuring armhole consistency is discussed.

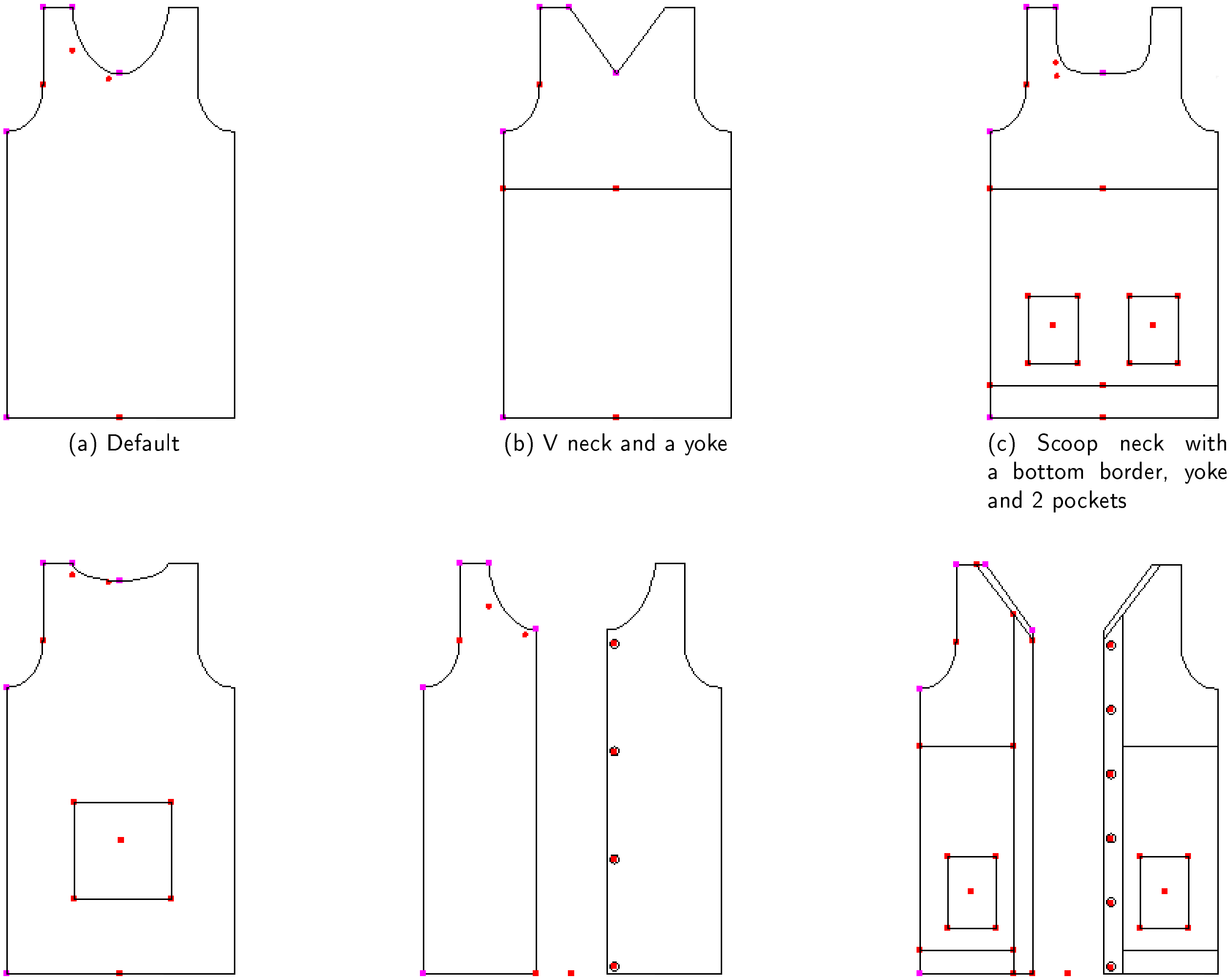
5.2.7.1 Instantiation

Making the user generate a sketch completely from scratch would be deemed to be unacceptably laborious, since cardigans and sweaters tend to have a similar structure. It is far more efficient to be given an object, to which small changes are required, than to start with nothing at all. Also, for the questionnaire to be useful in case based reasoning, there should be an approximate correspondence between the options in the questionnaire and the features in the sketch. Therefore, SEACOP generates a sketch from the questionnaire which is a useful starting point from which the designer can make their creative modifications.

As described previously, SEACOP's sketch instantiation algorithm takes a set of questionnaire responses as its input. Its output consists of a sketch in the *default size* (which is the smallest size that SEACOP is capable of working with). The sketch has a set of pieces (front, sleeve etc.), and the associated sketch features: shape points, move advisers and constrainers. The algorithms used are not particularly sophisticated and consist of conditional programming.

The complexity of the operation lies in the questionnaire, which contains at least 30 questions. Some of the questions do not affect the sketch, for example the choice of background stitch. However, other options (such as neck shape) have a substantial effect. Some combinations of options are not applicable, for example it is not possible to add a centre pocket to a cardigan since it is divided down the middle. Other options are highly interactive, such as the options for bottom border, front border, yoke and neck band; the composition of the move advisers and constrainers which are added depends on the combination of options chosen.

Figure 5.25 shows some example configurations of how the front may appear; each one arises from a



(a) Default

(b) V neck and a yoke

(c) Scoop neck with a bottom border, yoke and 2 pockets

(d) Slash neck and one central pocket

(e) Simple cardigan with round neck and 4 buttons

(f) Cardigan with v-neck, bottom border, neck band, front border, yoke, two front pockets and 6 buttons

Figure 5.25: Example combinations of options for the sketch of the front

Stretch	*Front*	*Back*	*Sleeves*
Armhole depth	✓	✓	✓
Straight part at the top			✓
Raglan			✓
Width at back of neck	✓	✓	
Neck depth	✓		

Table 5.5: Localised Stretches

different set of questionnaire options. Further examples are given in appendix D. An explanation of the conventions and heuristics used for neck shape and other design choices is given in chapter 4.

The designer may use the questionnaire to changes the range of sizes so that the smallest size is different from the *default size*. In this situation, the sketch is generated in the default size, then the resizing algorithm is invoked to transform the sketch into the smallest size in the range that the designer has specified.

5.2.7.2 Resizing

Commercially produced knitting patterns typically work with a range of sizes, since it is possible to produce garments from the same design to fit a wide size range of wearers. Therefore, SEACOP is capable of producing sketches which are consistent with each other, but correspond to different sizes of garment.

The instantiation algorithm generates a sketch in one size which the user is free to edit, subject to the restrictions mentioned earlier in this chapter. To ensure consistency, only the *smallest size* can be edited by the user; the other sizes are generated from this using the *resizing algorithm*. Whereas the sketch instantiation algorithm produces a sketch from questionnaire responses, the *resizing algorithm* produces a sketch from another sketch. The goal is that key measurements adhere to established guidelines for garment sizing (see section 4.3 for more information), with as little qualitative distortion in the shapes as possible.

The *resizing algorithm* first employs a *global uniform scaling*; the scale factor is simply the increase in bust or chest size. Thus, to scale from a 32" waist garment up to 34", the scaling would be $\frac{34}{32}$. Secondly, there are *localised stretches*, which ensure that the garment conforms to knitwear sizing guidelines. These stretches are applied to localised regions of certain pieces, as indicated in table 5.5. Within the region, points are subjected to the same scale factor, with the exception of the Bézier curve control points; these are moved in such a way that the character of the curve is not distorted.[18]

[18] In the same way as if they were being controlled by a Proportional Mover.

5.2.7.3 Symmetry and Sleeves

Symmetry Garments are symmetrical by default, so only one half of a symmetrical piece is generated by the sketch instantiation algorithm. The shape points in the other half are generated 'on the fly' by reflection. The vast majority of garments are symmetrical, so it is appropriate for symmetry enforcement to be the default option.

There are some exceptions to symmetry, as explained in section 5.2.4.3. Buttons are never reflected, and panels are sometimes not (depending on the user's preference and their geometry). Pockets are not reflected, but they are kept consistent by default with move advisers.

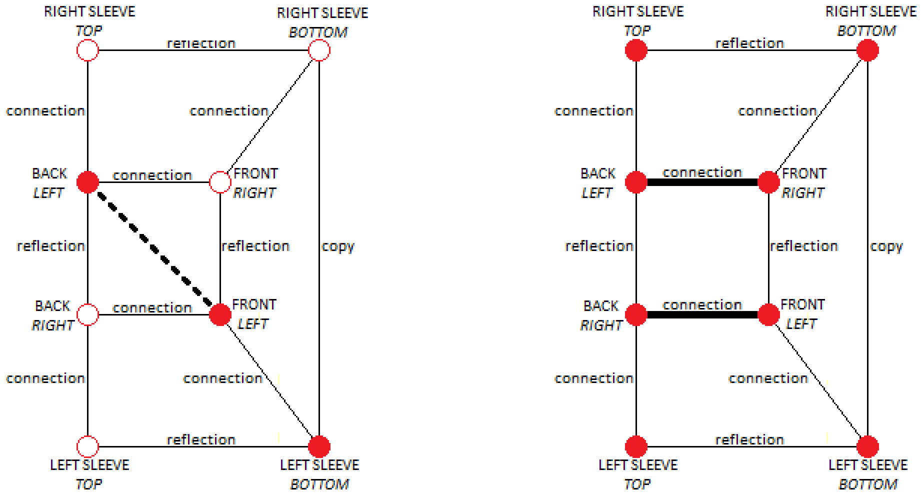
Symmetry can be “turned off” (as explained in section 5.2.4.4), giving the designer the freedom to create asymmetric garments in order to respond to fashion trends. When symmetry is turned off, the Shape Points in the “missing” half are generated by reflection, in the same way as a symmetrical piece, but crucially they are then are stored in the piece. Any sketch controls (move advisers or constrainers) are copied so that they involve the Shape Points on the “missing” half, and any parameters that are applicable to those sketch controls are obtained by reflection from the sketch control which it is copied from.

Identical Sleeves As explained in section 5.2.2.1, by default both sleeves are identical, and only the left sleeve is editable. The vast majority of patterns have identical sleeves so (as with symmetrical pieces) this is chosen as the default option. For the purposes of generating a knitting chart (see appendix B), the sketch for the right sleeve is generated 'on the fly' by copying and reflecting that of the left sleeve.

If the user opts to make both sleeves editable, then the sketch for the right sleeve is generated by copying and reflecting that of the left sleeve, including sketch controls. Once the sleeve is copied, maintaining consistency between the sleeves is the responsibility of the user. Therefore, if objects (such as rectangular panels) are required to be present on both sleeves in similar or identical positions, then the user will add them to the left sleeve before making the right sleeve editable, so that the objects will be guaranteed to be consistently placed at that point. This explains why making the right sleeve editable does not cause the sketch to be re-instantiated, since this would lead to a loss of any features that had been added (to the left sleeve) by the user.

Garment Relationships SEACOP stores the relationships between the pieces of the garment using a graph. A *place* is one half of a piece, i.e. the left or right of a body piece, or the top or bottom of a sleeve[19]. Two places may be involved in a *relationship*. The relationships graph consists of places as its vertices and relationships as edges.

[19] The places are described from the point of view of the user viewing the sketch on screen. For example, the highlighted regions in figure 5.27 belong to the *bottom* of the sleeve and the *left* of the body.



(a) Default garment (b) Fully editable garment

Figure 5.26: Garment Relationships Graph

The principal purpose of the relationships graph is for the *propagation of linked points*. In the default garment, the back-left and front-left places are linked, since those are the only editable places available (this is known as an *artificial* relationship). The effect of symmetry is to then make all four body places consistent. If either the front or the back are made asymmetric, then the linked points will be changed to include the newly editable place instead of the *artificial* relationship. The pairs of points that are to be linked in the newly editable place are obtained using the relationships graph, as described below.

If there is a relationship between two places, this means that a 1:1 mapping can be defined between Shape Points in one place and those in the other. A relationship may (or may not) be between two places that are linked via a set of linked points. In figure 5.26a, editable places are shown with a solid circle, and non-editable with a hollow circle. Relationships which involve *linked points* are shown in bold. There are four types of relationship:

- *Reflection:* where the sketch on one of the places is (or could be) obtained by reflecting that in the other; this occurs in symmetrical pieces (see the discussion of symmetry, above).

- *Copy:* where the sketch in one of the places is (or could be) obtained by copying that of the other; this happens when the user makes the sleeves non-identical (see the discussion about identical sleeves, above).

- *Connection:* where the two places are not linked by a *reflection* or *copy*, but nevertheless are physically sewn together when the garment is made.

- *Artificial:* where there is in fact no natural relationship but one had to be assumed in order to create linked points. This is shown by a thick dashed diagonal line in figure 5.26a.

The *propagation of linked points* occurs whenever either the front-right or back-right become editable.

The first stage is to perform a search for cycles[20] on the relationship graph. The object of the search is to find a cycle in which:

- one edge corresponds to a connection with linked points (the *source*)
- one edge corresponds to a connection which does not have linked points (the *sink*)
- all the vertices correspond to editable places.

If a suitable cycle is found, the linked points in the *source* are copied to those in the *sink*, using the 1:1 correspondences between the places. If the *source* was an artificial connection, this will be removed, i.e. the linked points objects will be deleted. For example, in the default garment, the user may decide to make the front asymmetric, hence the front-right place becomes editable. A suitable cycle is found: {front right}-{front left}-{back left}-{front right}. A new set of linked points is created between {back left}-{front right}, and those between {front left}-{back left} are removed.

Each relationship may also be associated with an affine transformation, so that SEACOP is aware of the geometric relationships that exist between the *places* when the sketch is realised as an actual garment.

5.2.7.4 Armhole Consistency

Armhole consistency is a soft constraint in SEACOP that is handled independently of the movement process. So that the body and sleeve pieces can be sewn together, the length of the sleeve head and armhole must be roughly equal. This ensures that the sketch corresponds to a design which is knittable and a garment which is usable.

The regions in the sleeve and body that must match are delineated by two specific points in the sketch, as shown by the bold lines in figure 5.27. In between those will be lines, Bézier curves and elliptical quadrants; the total lengths of these on the two pieces that must be equal.

SEACOP comes with the means to both detect and repair violations of armhole consistency. Table 5.6 shows an example of information which is available to the user, about the default garment. Each row of data represents a sleeve-body pairing that ought to be kept consistent. In a garment where all pieces are asymmetric (i.e. the sleeves are non-identical), there will be four rows, as per table 5.7.

In tables 5.6 and 5.7, the body place (1) refers to the region on the front or back, which can be resized automatically on user request. The ideal is that place (1) should be consistent with place (2), which will be a sleeve. Sometimes, place (2) is not available, due to piece symmetry; in which case the consistency algorithm will work with the equivalent place, which is (3). The deviation between the lengths of places (1) and (3) is given by:

$$d = 100\frac{|l_1 - l_3|}{l_3}$$

[20] The algorithm for finding cycles is not sophisticated but the graphs are fairly small so the issue of combinatorial explosion is a manageable one.

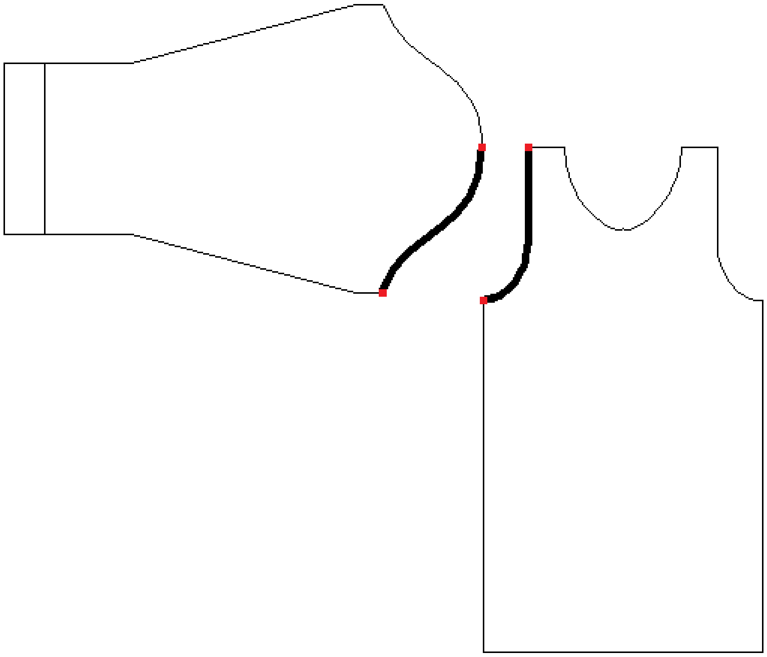


Figure 5.27: Regions that must have a consistent length

(1) Body place : resizeable		*(2) Sleeve place: connected to*	*(3) Sleeve place*		*Deviation %*
Place name	*Length, cm*		*Place name*	*Length, cm*	
front, left	19.6	*(1)*left sleeve, bottom	left sleeve, bottom	19.6	0.004
back, left	19.6	right sleeve, top	left sleeve, bottom	19.6	0.004

Table 5.6: Consistency information for default garment

The regions which are being measured consist of lines, Bézier curves and elliptical quadrants. The length of a Bézier curve is estimated by approximating it as a series of line segments, and summing the length of these. The length of an elliptical quadrant is estimated as:

$$length = \begin{cases} \frac{\pi}{4}\left(a+b\right)\left(1+\frac{3h}{10+\sqrt{4-3h}}\right) & if\, h < 0.16 \\ \frac{\pi}{4}(a+b)\left(\frac{4}{\pi}\right)^{h} & if\, h \geq 0.16 \end{cases}$$

where a is the semi-major axis and b is the semi-minor axis of the underlying ellipse, and:

$$h = \frac{(a-b)^2}{(a+b)^2}$$

The formula for $h < 0.16$ is from Ramanujan [155], and that for $h \geq 0.16$ was devised by Zafary [156]. Both formulas are approximations but the Zafary formula appears to be more accurate when the ellipse is flattened and the Ramanjuan one when it is not flattened.

The equalisation algorithm works by performing a localised stretch of the armhole region. A binary

(1) Body place : resizeable		*(2) Sleeve place: connected to (1)*	*(3) Sleeve place*		*Deviation %*
Place name	*Length, cm*		*Place name*	*Length, cm*	
front, right	21.0	right sleeve, bottom	right sleeve, bottom	19.6	7.5
front, left	19.6	left sleeve, bottom	left sleeve, bottom	19.7	0.8
back, left	19.6	left sleeve, top	left sleeve, top	19.6	0.0
back, left	17.4	right sleeve, top	right sleeve, top	19.3	9.4

Table 5.7: Consistency information for fully editable garment which has been edited to introduce some asymmetry

search is used to optimise the scale factor used in the stretch, the search terminates when the body region has approximately the same length as the sleeve region, within an acceptable tolerance.

Through the armhole consistency mechanism, the designer can be confident that the sleeve and armhole are compatible. Through the use of linked points, the length of the body pieces can also be kept consistent. Thus the designer is assured that the sketch they are producing is realisable as a functional garment.

Chapter 6

Representing and comparing designs

This chapter discusses both *knowledge representation* and *similarity*. The two concepts are related since any measure of similarity must take into account the knowledge representation. When determining how to represent knowledge, the main consideration (apart from the nature of that knowledge itself) should be the use that knowledge will be put to. In SEACOP there are two uses: for case based reasoning (CBR) and computer-aided design (CAD). The two uses are interlinked, as the CAD functionality (which is discussed in chapter 5) is used to view and edit the output of the CBR.

Thus, the difference between case representation in SEACOP and that in other CBR systems (see section 2.2.1) is that the requirements are partly determined by the needs of CAD. Also, the representation of cases in SEACOP is probably more complex than in many other CBR systems, since it includes features from different stages of the design process.

Zeleny [157] explained knowledge by analogy: *data* is like the atoms and molecules in food ingredients, *information* is like the ingredients themselves, *knowledge* is like a recipe, and *wisdom* is the insight into whether (or when) to make the food. From this analogy, it seems reasonable that the representation described in this chapter is a form of knowledge, since it contains not just the structure of a garment, but also its constraints.

As discussed in section 2.2.2.2, similarity is arguably the most important concept in CBR, since it is assumed to be a guide to adaptability [4]. If we have a query case (Q) and a retrieved case (R), then we assume that the similarity of Q to R is a predictor of how likely it is that R can be adapted into Q. Thus, the success of adaptation is partly dependent on having an effective similarity algorithm; such an algorithm is explained in section 6.2, which describes how *questionnaires can be compared*.

Section 6.3 discusses the *comparison of sketches*. It is more difficult to construct an algorithm to compare sketches than questionnaires. Such an algorithm can be used as an objective assessment of the success of adaptation, by comparing the output of CBR with the goal or finished version of the garment.

6.1 Knowledge Representation

This section begins with a 'high-level' discussion of the scope of the knowledge, and the features of a good representation. Finally, section 6.1.3 discusses how the knowledge is represented in practice.

6.1.1 Levels of detail

The knowledge represented by SEACOP is in three levels of detail: questionnaire, sketch and chart. The sketch is derived from the questionnaire, and is more detailed. Similarly, the chart is the most detailed detail, and is derived from the sketch.

A garment consists of *pieces*; each *piece* corresponds to a separate piece of fabric. The options for types of piece are chosen from the set {front, back, left sleeve, right sleeve, collar, hood}. Each sketch and chart is associated with a particular piece.

A garment must have a questionnaire. The questionnaire does not relate to a specific piece but to the garment as a whole. It is possible for individual questionnaire options to affect only one piece, but some affect more than one, e.g. the armhole style affects the front, back and both sleeves.

- The *questionnaire* consists of high-level choices made by the user, early on in the design stage. For example, the user will choose the type of neck, style of armhole and length of sleeve. Many of these options are selected from a discrete list of choices; others are numeric or textual values.

- The *sketch* consists of a series of points; these have a location in the 2-D Euclidean plane. The points represent the shape of a piece. Additional data is also associated with the points, such as a textual description.

- The *chart* consists of knitting stitches, arranged in rows. Each stitch is unique by virtue only of its location in the garment; however the stitch has a type. The number of types of stitch is fairly small (circa 40).

Since the structure of the questionnaire simply approximates to a series of attribute-value pairs, it does not warrant detailed discussion. The important features of the questionnaire are listed in section 6.2.2.3. Some other attributes are present but are not listed, for example a short textual description of the garment; these other attributes do not have an effect on the sketch and are for the designer's convenience only.

The details of the representation of the sketch is discussed in section 5.2.5. A piece consists of an ordered list of *Shape Points*; these define key points in the sketch such as the ends of line segments, and specify the curves. *Move advisers* and *constrainers* are also an integral part of the structure of a piece; these implement soft constraints such as corners being right angles.

Since the case based reasoning functionality developed as a part of this project does not involve knitting charts, they are discussed in appendix B.2.7.

6.1.2 Features of a good representation from the domain perspective

6.1.2.1 Heterogeneity: coping with differences in composition

The composition of a *completed* garment is heterogeneous, in several respects:

- At the highest level, a garment must have a front and back, but the other pieces are optional. A garment will have between zero or one pieces from the subset {collar, hood}. The constraints on the subset {left sleeve, right sleeve} are explained in section 6.1.2.2.
- The questionnaire may be described as a set of attribute-value pairs. However, not all attributes are present: the applicability of some is dependent on the values of others. For example, the choice of front border stitch is only relevant in cardigans. Also, the range of applicable values for one attribute bay be affected by the value of another attribute. For example, a cardigan cannot have a centre pocket.
- The set of points that comprise the sketch is different from garment to garment. The set of points that are created automatically is determined by the questionnaire (as described in section 5.2.7.1) therefore not all points may be present in every sketch, and the locations may be different depending on the options chosen in the questionnaire. Also, users can edit the sketch by moving the existing points and adding additional points (as described in section 5.2.4.2).
- The chart is generated from the sketch (as described in appendix B), however the user can edit the chart; hence its composition is not completely predetermined.

As well as completed garments, SEACOP needs to be able to represent *incomplete* garments at various stages of the design process. Therefore, it is optional for a piece to have a sketch. If the piece has a sketch, it may also optionally have a chart. A piece will not have a chart without an accompanying sketch.

6.1.2.2 Symmetry

Symmetry is a soft constraint in knitwear design which is commonly applicable. Symmetry is a feature of two levels of the representation: questionnaire and sketch. In the questionnaire, the user is asked whether the sleeves are symmetrical. If the answer is yes, then the right and left sleeves are mirror images of each other; in this scenario, only one sleeve needs to be represented, as the other can be derived from it. For consistency, the left sleeve is always present in a sleeved garment, as per table 6.1.

Symmetry is also a feature of most garments at the sketch level. In a symmetrical sketch, only half of the points need to be represented; the positions of the others can be derived. One point is defined as the origin, and this is used to define the axis of symmetry. Front and back pieces have a vertical axis,

Questionnaire option		*Piece composition*	
has sleeves?	*sleeve symmetry?*	*left sleeve*	*right sleeve*
no	n/a	absent	absent
yes	no	present	present
	yes	present	absent

Table 6.1: Sleeves and symmetry

and sleeves a horizontal axis of symmetry. If the user requires an asymmetrical garment then this fact needs to be recorded and all of the points will feature in the sketch representation.

6.1.2.3 Flexibility: coping with changes in requirements

The representation will need to have some flexibility at all three levels: questionnaire, sketch and chart.

A good representation will be flexible enough to cope with changes in requirements, with modifications to the structure being as minimal as possible. Some of the attribute value pairs in the questionnaire contain data that could be used to generate a textual preamble for the knitting pattern. The format of this may change if the user's requirements change, and thus the questionnaire format must not be too rigid.

In addition to the points, the sketches have *sketch controls*, which enforce soft constraints. There are different types of sketch control; the controls affect a variable number of points and also contain parameters. So, sketch controls are not simply a type of functionality, they contain knowledge which must be represented somehow. The sketch controls are added when the sketch is created, depending on the options in the questionnaire; thus, each garment has a different set of sketch controls. If the user decides to 'turn off' a sketch control (as explained in section 5.2.4.4), it is removed from that set. There are two options for representing sketch controls:

1 The representation of the sketch controls could be separate to that of the garment, and if each control is given a unique identifier then the sketch simply needs to record which controls are active.

2 Alternatively, the sketch controls can be an integral part of the sketch representation.

Option 2 gives the greatest flexibility. It would make it easier to extend the system (if required in future) for users to be able to alter the parameters of the sketch controls, or even to create their own sketch controls dynamically. This would permit the automatic enforcement of constraints which were unexpected at the time SEACOP was created.

More limited flexibility is required at the chart level. As discussed in section 4.4, the chart is composed of stitches: a large number of types of stitch are theoretically possible, but in practice a relatively small number are in regular use. To try to represent everything that is possible to achieve with a knitting needle would be virtually impossible, and would unnecessarily over-complicate the system, since the majority of stitches would not be used.

However, the stitch types that are employed are used in many different garments. Therefore, it makes sense for the representation of the types of stitch to be separate from that of the chart. All the chart needs to record is which types of stitch are used where; those stitches are defined elsewhere. The representation of stitch types needs to be flexible, however. It must be possible to define new types of stitch, should the existing ones be insufficient. In particular, new types of cable stitch are not uncommon.

6.1.2.4 Redundancy

The inclusion of three levels in the representation may appear to be redundant: if the sketch is derived from the questionnaire, then why is the questionnaire not then "thrown away" ? Redundancy is avoided in other ways, for example when garments are symmetrical (see section 6.1.2.2), so why not be consistent and avoid redundancy altogether?

In fact, there are good reasons why the representation needs to include all three levels. The questionnaire contains information unconnected to the sketch, such as a description of the yarns used; this information would be essential if the system were to output knitting patterns. The questionnaire is also needed for defining the measure of similarity in CBR (see section 5.2.2). Furthermore, the sketch is needed to act as a measure of success of CBR (see section 6.3). The sketch and chart can be edited by the user, which means that it is not necessarily possible to reverse the process and obtain a unique questionnaire from a sketch, or a sketch from a chart.

6.1.2.5 Persistence

The representation must allow conversion to persistent means, i.e. a file on a computer's hard disk. Therefore there are two implementations of the representation: a *volatile* one, and a *persistent* one. When the garment is being edited by the user or involved in CBR, it is the volatile version that is being read or manipulated. It must be possible to convert from volatile to persistent, and vice versa, without loss of integrity. It is advantageous (but not absolutely essential) if the persistent representation fulfils these two criteria:

- *Readability*. If the persistent data is stored in a format that is intelligible to a human, this can facilitate debugging.

- *Compactness of charts*. The questionnaire and sketch do not contain a lot of duplication. In contrast, charts consist of thousands of knitting stitches, and there is extensive duplication. For example, many rows of knitting consist of several (30-100) copies of the same "stitch", and when a part of the piece is rectangular it is often the case that the instructions for a row are duplicated many times.

6.1.3 Implementation

To summarise the discussion in the previous sections, the knowledge to be represented is complex as it is multi-level and contains both structure and constraints. Both volatile (in-memory) and persistent storage of the knowledge must be facilitated. SEACOP requires a flexible way of representing garment knowledge which supports both CAD and CBR.

When considering the choice of implementation, compartmentalising the problem into a CAD system and a CBR system is one solution, but it could lead to significant duplication of effort. Therefore, a systemic approach is taken and the problem considered holistically.

6.1.3.1 Volatile (in-memory)

The easiest decision to make is the means of implementing CAD. The criteria for purchasing an existing system were:

1 Available at low (or no) cost.

2 Functionality is for hand knitting (not machine knitting or general CAD).

3 Ability to integrate with a CBR system (either the CAD system must be open-source and extensible, or an API provided).

If general CAD functionality had been required, it may have been possible to link this to CBR, as this has been done previously (e.g. [158]). However, no existing system fulfils the above criteria; therefore the task was to create one.

In order to implement CBR, CBR shells were considered. The most natural candidate would be jCOLIBRI (which was discussed in section 2.2.5.1) because it is extensible and flexible. However, due to the specialised nature of the functionality required (as discussed in section 6.2 and chapter 7), it seems likely that jCOLIBRI would add little value to the project. This was confirmed by a personal conversation with a member of the GAIA group at Complutense University, who developed jCOLIBRI. Much of the functionality would have to be written using bespoke program code, which reduces the benefit that would be gained significantly. Also, ensuring that the existing program interfaced with jCOLIBRI would be an additional burden. The software engineering aspects of the project would be constrained by the needs of jCOLIBRI, and the benefits are likely to be minimal since much of the functionality would be bespoke anyway. jCOLIBRI is a powerful tool and a significant contribution to CBR research, but it is not an appropriate means of delivering the functionality discussed here.

So, the requirements of CBR and CAD suggest the construction of a bespoke piece of software, presumably written in a high-level language to aid productivity. Java was chosen because it is freely available, comes with access to free integrated development environments (IDEs) which can quickly create user interfaces, and has a built-in library for functionality such as mathematical functions.

6.1.3.2 Persistent

Choice of technology

In order to implement the persistent storage of the knowledge, several options were considered:

- *Databases*: have the advantage that concurrent access is well managed; however this is not a feature of the problem in hand. To store an entire garment in a relational database may have a significant performance overhead, due to the joins required. Although software is available to facilitate the editing and viewing of data in databases, only one table is typically seen at a time. It would be very difficult to construct a single view of an entire garment.

- *Ontologies*: these are flexible and powerful in that they represent the semantics between terms, however this functionality is not required. Ontologies were rejected as the time spent in defining the concepts would not be well spent, and as with databases it might be difficult to get a single view of an entire garment.

- *Object serialisation*: Java comes with a facility to serialise objects so that persistent storage is achieved with minimal effort on the part of the programmer, making this an apparently very attractive option. However, the resulting files are binary (not intelligible to a human). Also, experiments showed that Java's object serialisation failed with knitting charts; the implementation was memory-hungry and knitting charts contain thousands of objects.

- *CSV or text files*: these have the advantage that they are very flexible, since the software engineer or knowledge engineer decides the structure. They can be read in or written out from a program very quickly. They can also be modified manually using text editors. However, the files are not self-describing and have no in-built structure; it is up to the programmer to define a structure. It would be very easy to corrupt a text file using an editor, and this corruption would not be apparent until the program accessed the file.

- Extensible Markup Language (XML) was selected as the best means of implementing persistent storage, since it offers the key advantage that it is possible for a human to read the contents of an XML file, in one place, and understand the data (it is self-describing). Also, XML offers some protection from corruption in that it is easy to check if an XML file is well-formed, without having to access the SEACOP. Thus, the use of XML facilitates testing and debugging.

Interfacing XML and Java

Three technologies for interfacing Java and XML were evaluated:

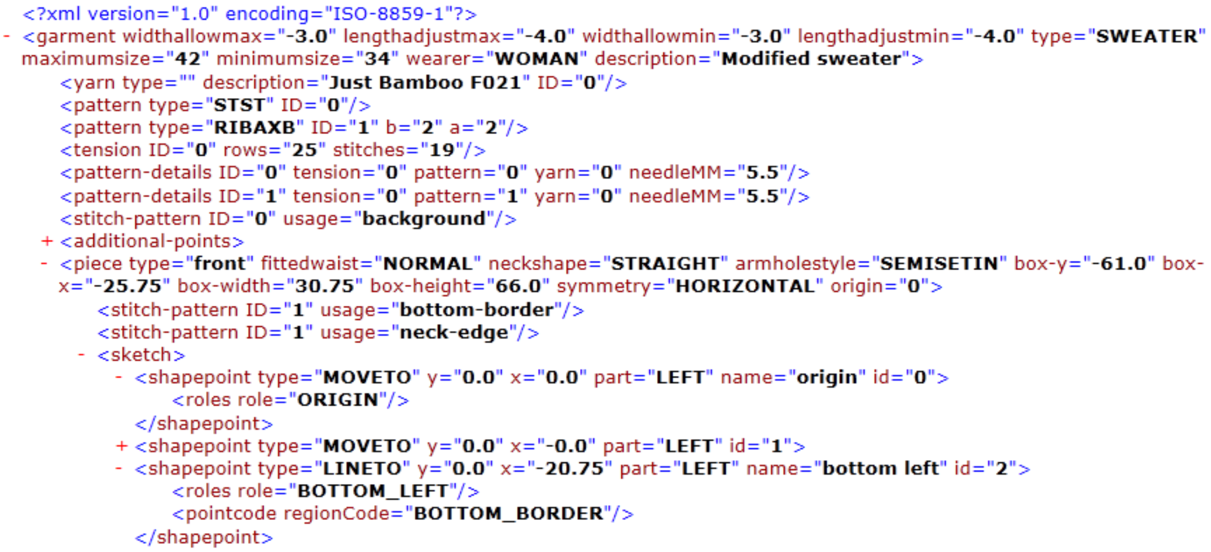
- *DOM* (Document Object Model) - this reads the entire file into a special data structure in memory. The program can then access all or any part of this data structure, in any order.
- *SAX* (Simple API for XML) - this parses the file and as each node is read, it raises an event which the programmer can trap. Although not explicitly specified, it is the convention that SAX parsers read the file from beginning to end, working forwards.
- *JAXB* (Java Architecture for XML Binding) - this allows the user to define an XML Schema, then automatically generates Java classes to hold the data in persistent storage. Reading to and from those classes is then done when required by the program, with very little coding required by the programmer.

All three technologies were applicable. The disadvantage of JAXB is that it approaches the whole software development problem primarily from the point of view of persistent storage. If the schema is changed, then the data classes will change and any other parts of the program that references these classes could be incompatible, causing compiler errors. The change cannot be tested until those errors are fixed. However, SEACOP was not developed in this way: iterative development was used, with the user interface and algorithms completed first, and then finally persistent storage. When developing a computer-aided design program, it is more intuitive to see how features look first, before worrying about how their data would be saved in a file. JAXB automates some aspects, but it causes the programmer to lose some control over part of the code.

SAX was chosen since it is faster and requires less memory than DOM. Also, a decision was taken to only read in whole garments from files; thus, the advantages of DOM were not applicable. The decision to choose SAX came at a price: out of approximately 90000 lines of code in SEACOP, 2275 lines were directly concerned with writing garment to XML or reading them from XML.

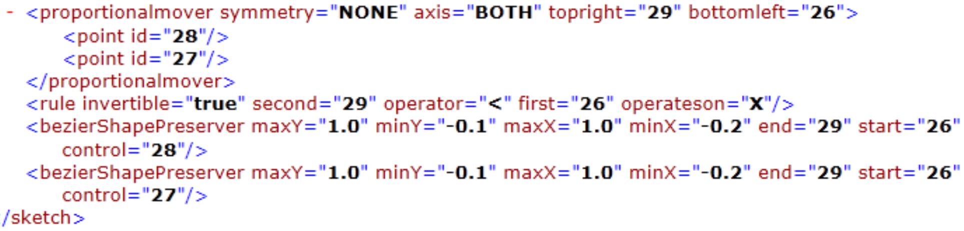
Example of XML

Figure 6.2 shows a sample portion of an XML file for a garment. The file begins at the garment node with questionnaire information being specified. Then, there are some definitions: of yarns, tensions and stitch patterns. As with many features, these are given a unique identifier to enable the object relationships to be created accurately when the file is read. In the piece node, the start of the specification of the front piece is shown; this also has some associated questionnaire information. Then, the patterns used in the front are defined: the bottom border and neck edge use pater ID 1, which has been defined previously as a 2x2 rib. Then, in the sketch node, there follows a list of shape points.



```
  <?xml version="1.0" encoding="ISO-8859-1"?>
- <garment widthallowmax="-3.0" lengthadjustmax="-4.0" widthallowmin="-3.0" lengthadjustmin="-4.0" type="SWEATER"
  maximumsize="42" minimumsize="34" wearer="WOMAN" description="Modified sweater">
    <yarn type="" description="Just Bamboo F021" ID="0"/>
    <pattern type="STST" ID="0"/>
    <pattern type="RIBAXB" ID="1" b="2" a="2"/>
    <tension ID="0" rows="25" stitches="19"/>
    <pattern-details ID="0" tension="0" pattern="0" yarn="0" needleMM="5.5"/>
    <pattern-details ID="1" tension="0" pattern="1" yarn="0" needleMM="5.5"/>
    <stitch-pattern ID="0" usage="background"/>
  + <additional-points>
  - <piece type="front" fittedwaist="NORMAL" neckshape="STRAIGHT" armholestyle="SEMISETIN" box-y="-61.0" box-
    x="-25.75" box-width="30.75" box-height="66.0" symmetry="HORIZONTAL" origin="0">
      <stitch-pattern ID="1" usage="bottom-border"/>
      <stitch-pattern ID="1" usage="neck-edge"/>
    - <sketch>
      - <shapepoint type="MOVETO" y="0.0" x="0.0" part="LEFT" name="origin" id="0">
          <roles role="ORIGIN"/>
        </shapepoint>
      + <shapepoint type="MOVETO" y="0.0" x="-0.0" part="LEFT" id="1">
      - <shapepoint type="LINETO" y="0.0" x="-20.75" part="LEFT" name="bottom left" id="2">
          <roles role="BOTTOM_LEFT"/>
          <pointcode regionCode="BOTTOM_BORDER"/>
        </shapepoint>
```

Figure 6.1: Sample portion of the XML file for a garment



```
- <proportionalmover symmetry="NONE" axis="BOTH" topright="29" bottomleft="26">
    <point id="28"/>
    <point id="27"/>
  </proportionalmover>
  <rule invertible="true" second="29" operator="<" first="26" operateson="X"/>
  <bezierShapePreserver maxY="1.0" minY="-0.1" maxX="1.0" minX="-0.2" end="29" start="26"
    control="28"/>
  <bezierShapePreserver maxY="1.0" minY="-0.1" maxX="1.0" minX="-0.2" end="29" start="26"
    control="27"/>
</sketch>
```

Figure 6.2: Sample portion of an XML file showing move advisers

Move advisers are an integral part of the XML file for a garment, as illustrated in figure 6.2. This shows a proportional mover, two Bézier shape preservers, and a rule being defined. The integers are the IDs of shape points which were defined elsewhere.

Charts are also an integral part of the XML file. A simple form of run-length encoding is used when a stitch occurs a number of times consecutively in a row; only the type of stitch and an integer corresponding to the number of occurrences needs to be stored.

Representing knowledge shared between garments

As well as the individual garments, SEACOP needs to store two additional types of data: stitches and stitch patterns. Storage of stitches is the simplest: these are all kept in one comma separated values (CSV) file. Before SEACOP undertakes CBR or CAD functionality, it first loads this file into memory. The data in the file roughly corresponds to the table of stitches in appendix A. Each stitch has a unique string identifier. The file is extremely compact and is loaded quickly into memory. Although CSV lacks the elegance of XML, this file rarely requires editing so this is not important.

Stitch patterns are stored as XML files. As with stitches, stitch patterns are shared between garments, so it is appropriate to store them separately. Three methods are used for storing stitch patterns:

- Standard patterns are those shown in the table in appendix A, e.g. “rice stitch”. There are a small number (8) of standard patterns, they have very small knitting charts, and they are used very commonly. Therefore, these patterns are 'hard-coded', i.e. defined using literals in the Java code.

- Ribs are parametric and have a knitting chart which is very simple to create algorithmically. An example of a 2x4 rib is shown in appendix A. Therefore, there is no need to store rib patterns and they are generated on demand.

- Bespoke stitch patterns, i.e. those created by the user (as described in section 5.2.3.2) are stored as CSV files. The file consists of a header row which specifies which columns of stitches form the repeat period. After the header row, there is a matrix of textual symbols, each one consisting of the unique identifier of a stitch. The rows and columns in the CSV file correspond directly to those in the knitting chart. This is a compact and efficient way to store the stitch pattern data. The path and file name of the CSV file is used as an identifier for the pattern, in the XML file for the garments that use it.

This chapter has explained how the representation of garment knowledge has been implemented after carefully considering the complexities of knowledge itself, and the requirements of the system that will use it. The representation chosen is efficient and reliable, and supports both the needs of CBR and CAD.

6.2 Questionnaire Similarity

6.2.1 Similarity for efficient retrieval in CBR

In the discussion of questionnaire similarity, Q refers to the query case and R to a retrieved (or potentially retrieved) case. There are two types of similarity between Q and R:

- It is essential that an indication of *relative similarity* is provided. If we have a query case Q, and two other cases R_1 and R_2, then the algorithm must be able to judge the relative similarity of R_1 and R_2 to Q. Thus, if we have a case base (R_1, ... R_n) then the case (or subset of cases) with a maximal similarity to Q can be retrieved.

- *Absolute similarity* is not essential but important: a bounded numeric score for the similarity of R to Q indicates if R is a close match to Q, or a not-so close match. Of course, if the algorithm outputs an absolute similarity, then this is sufficient to provide relative similarity.

As stated above, it is important that the similarity is bounded, and for convenience we define those bounds to be [0,1]. We refer to similarity rather than distance since the former is more intuitive in this problem domain. Similarity is the opposite of distance:

$$S(Q, R) \equiv 1 - D(Q, R)$$

6.2.1.1 Why questionnaires are compared

It is common practice in CBR to index cases in order to improve the performance of retrieval (as discussed in section 2.2.2.1). However, this is not necessary in SEACOP since the questionnaire part of a case acts as an inbuilt index[1]; it avoids the need to artificially construct one. Since the sketches are derived from the questionnaire, we know that the questionnaire data is *relevant* to garment similarity. Thus, the need to artificially construct an index is avoided. Comparing questionnaires is an *efficient* process since a questionnaire is a relatively small data structure that mostly consists of discrete attributes chosen from a range of values. This efficiency means that we can use a simple retrieval algorithm; complex optimisations are not required. This is useful since SEACOP does not work within a metric space, which is an assumption required for many such optimisations (e.g. fish and shrink, see section 2.2.2.2).

In contrast, comparing sketches is complex, for a number of reasons, for example the sketches are not polygons and are invariant under translation: see section 6.3.1 for an explanation of sketch comparison. Comparing charts is likely to be a particularly complex and slow process, since charts typically consist of several thousand knitting stitches. Also, if CBR is actually used to create knitting patterns, then only the questionnaire is likely to exist at the start of the process, so the sketch and chart are unlikely to be available.

6.2.1.2 Similarity function defined

The conditions necessary for a metric are classically cited [159] as:

1 *non-negativity*: $D(Q, R) > 0$

2 *identity*: $D(Q, R) = 0 \iff Q = R$

3 *symmetry*: $D(Q, R) = D(R, Q)$

4 *triangle inequality*: $d(Q, R_1) \leq d(Q, R_2) + d(R_2, R_1)$

The *similarity function* defined here is not a metric [97], but it satisfies the non-negativity and identity conditions. *Non-negativity* is a consequence of our choice of [0,1] as bounds. *Identity* is important, otherwise what can be more similar to an object than itself?

Symmetry does not hold in our domain, since we have heterogeneous cases. In general, there are several places in the questionnaire with optional features. In each of these places, the similarity algorithm must incorporate two factors:

[1] The questionnaire is not literally a form of index as it does not provide a means of excluding cases from the retrieval process (as per section 2.2.2.1). However, since it facilitates an efficient similarity measure, it removes the need for indexing.

- Whether there is agreement on the presence of the feature in Q and R, and
- If the feature is present in both Q and R: the similarity of the feature.

For example, if a query case Q has sleeves, then a sleeveless case will have a similarity dependent on the complexity of sleeves. The reverse is not true; if Q is sleeveless then when compared to an existing case with sleeves, the complexity of the sleeves is irrelevant [14]. Note that the situation with heterogeneous cases is not the same as with missing data. It is not appropriate to use statistical methods (such as substitution with the mean or modal value) to impute values which are not applicable.

6.2.1.3 Weighted sum approach

The knitwear domain is characterised by not being associated with an inherent similarity function. There is no effective and purely objective way to assess the similarity of two garments. There is no practical way, for example, to represent a garment as a graph (in order to utilise a transformation based measure such as the edit distance). Also, due to issues of dimensionality, retrieval methods that utilise the Pareto front or require extensive training (such as random forests) are not practical.

However, the composition of a questionnaire, with its attribute-value pairs, naturally lends itself to a simple weighted sum algorithm. Such an algorithm takes localised similarity scores, weights them to reflect their relative importance, and sums them to give a single overall similarity score.

$$S = \frac{\sum_{i=1}^{n} w_i s_i}{\sum_{i=1}^{n} w_i}$$

S refers to the overall similarity score, w_i refers to a weight associated with a local similarity value and s_i is the local similarity value.

6.2.2 Calculating the similarity

As explained above, the similarity function will be an asymmetric weighted sum approach which compares the questionnaires in the two garments. The major challenge in this is in obtaining the weights, and the way in which this challenge has been overcome is explained below.

6.2.2.1 Obtaining the weights

The primary disadvantage of an approach involving weights is that users and domain experts typically find it difficult to provide them. Many people are uncomfortable with estimating a numerical value which correlates to the importance of a particular feature. Therefore, our approach is to ask users to rank the features in order of importance, which is something that users tend to find more intuitive.

Description	*Function*
gentlest	$(n+1)^{\log_m(s+1)}$
gentler	$\frac{sm}{sm+n(1-m)}$
gentle	$m^{\frac{n}{s}}$
regular	$1+\frac{n}{s}(m-1)$
steep	$(1-m)\sqrt{1-\frac{n}{s}}+m$
steeper	$(1-m)\log_{s+1}(s+1-n)+m$
steepest	$1-\frac{n(1-m)}{s(s+1-n)}$

Table 6.2: weight generation functions

Firstly, the user divides the features into three groups. The *priority group* and *irrelevant group* are unsorted, but the features in the *ranked group* are arranged in order of importance by the user.

- *Priority group*: these features are all given the maximum weight.
- *Ranked group*: these features are given a weight in the range (0,1) as determined by their rank and the *weight generation function* (see below).[2]
- *Irrelevant group*: these features are all given a weight of zero.

The user is asked to choose the weight of the least important feature in the ranked group. Also, they must choose the *weight generation function*. The options are given in table 6.2, where:

- m is the weight of the least important feature in the ranked group.
- s is the number of features which are not in the irrelevant group, and which have a lower importance than another feature.
- n is an integer which reflects the relative importance of the feature.

In all situations, $0 \leq n \leq s$:

- If the priority group is empty, then n is the position of the feature within the ranked group; the most important feature has $n=0$, the next $n=1$ and so on. s is the size of the ranked group.
- Otherwise, $n=0$ for the features in the priority group; $n=1$ for the most important feature in the ranked group, and so on. s is the size of the ranked group minus one.

The weight generation functions have been given descriptive names which refer to their gradient when n takes large values. They have been chosen because they exhibit the following characteristics[3]:

- They are all defined when s is positive, $n \leq s$ and m is in the range (0,1).

[2] Or in the range (0,1] if the priority group is empty.

[3] The functions are all continuous but this is of lesser importance since n only takes integer values.

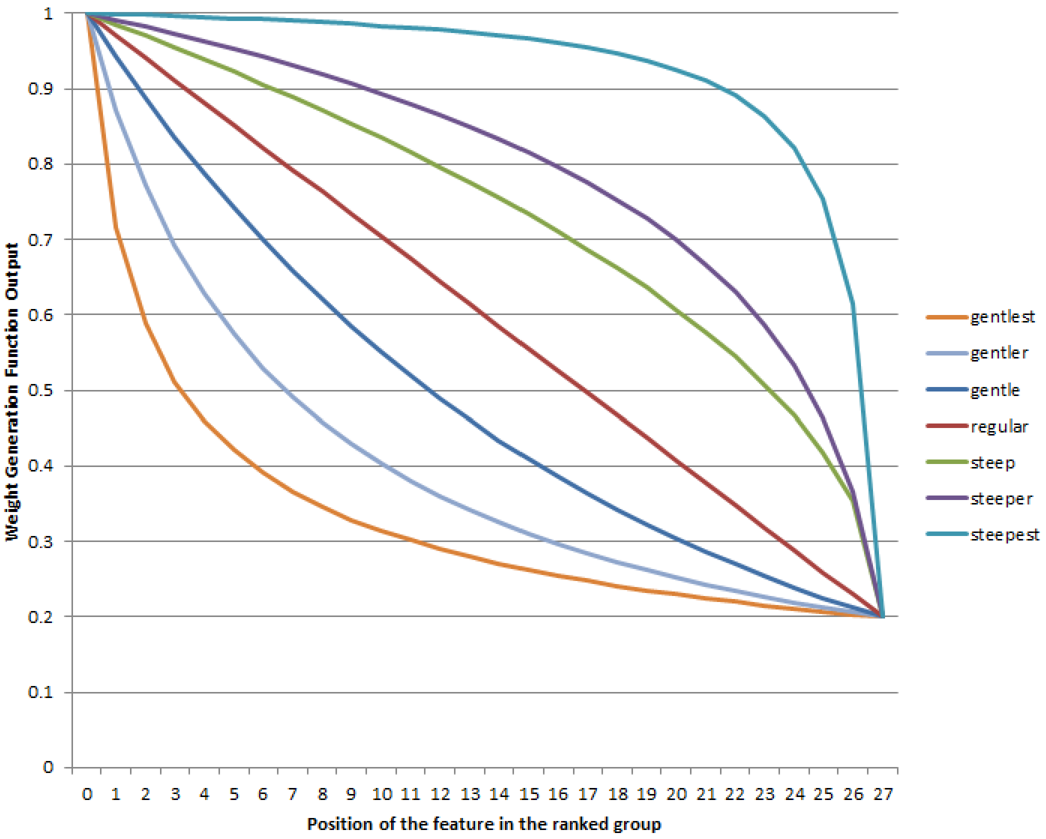


Figure 6.3: Example output from the different weight generation functions

- If $n = 0$ then the output is 1.
- If $n = s$ then the output is m.
- They are all strictly decreasing; it is important that within the ranked group, a lower ranked feature should always have a lower weight than a higher ranked feature.

Figure 6.3 shows example output from these functions when $s = 28$ and $m = 0.2$ [14, 97].

6.2.2.2 Assessment of similarity

The overall similarity of a query case relative to an existing case is given by:

$$S(Q, R) = \frac{\sum_{i=1}^{n} w(i)s(Q, R, i)}{\sum_{i=1}^{n} w(i)\delta(Q, i)}$$

Where:

- $w(i)$ is the weight of the ith feature, obtained as described in section 6.2.2.1.
- $s(Q, R, i)$ is the local similarity.
- $\delta(Q, i)$ is the relevance of the ith feature to the query case, as explained below.

Relevance

The term $\delta(Q,i)$ relates to the relevance of a particular feature.

$$\delta(Q,i) = \begin{cases} 1 & \text{if the ith feature is relevant to Q} \\ 0 & \text{otherwise} \end{cases}$$

For example, the relevance of the stitch pattern used for the cuff will be:

- 1 (relevant) in a query case with cuffs
- 0 (irrelevant) in a query case with no cuffs (or no sleeves).

If a particular feature is irrelevant to a particular query case, then it is necessary that the local similarity calculation for that feature results in zero, regardless of which case Q is being compared with, i.e.:

$$\delta(Q,i) = 0 \implies s(Q,X,i) = 0$$

If features are relevant, then the assessment of the local similarity depends on the nature of those features. Several types of local similarity function are used in SEACOP: each of these is described below.

Real number

The similarity between real numeric values is assessed using:

$$s(a,b) = \max\left(1 - \ln(|a-b|+1), 0\right)$$

This function is chosen because it is smooth, monotonic, and has the codomain [0,1]. Also, it does not rely on either a or b being bounded.

Logical biconditional

Logical biconditional is used when an option is either true or false in both cases.

$$s(b_Q, b_R) = \begin{cases} 1 & \text{if } b_Q \iff b_E \\ 0 & \{\text{otherwise}\} \end{cases}$$

For example, one particular feature is the presence of the bottom border:

- If both query case and existing case have a border then the local similarity is 1,
- If neither have a border then the local similarity is 1,
- Otherwise, the local similarity is 0.

Description	*Likert Score*	*Similarity*
very different	1	0.00
quite different	2	0.25
some similarities	3	0.50
many similarities	4	0.75
identical	5	1.00

Table 6.3: Likert score translations

	Entire front	*Top part only*	*Bottom part only*	*Middle only*
Entire front	5	2	2	3
Top part only	2	5	1	2
Bottom part only	2	1	5	2
Middle only	3	2	2	5

Table 6.4: Example of the use of a Likert matrix for symbolic features

Equality

Equality similarity is used when a contribution to the similarity is only appropriate if there is an exact match between two attributes.

$$s(v_Q, v_R) = \begin{cases} 1 & \text{if} \, v_Q = v_R \\ 0 & \{\text{otherwise}\} \end{cases}$$

Equality similarity will be used:

- For a symbolic attribute, where there are only two choices of symbol, e.g. {sweater, cardigan}.
- For an attribute which takes integer values, where the real number comparison (as described above) is not appropriate. For example, the number of buttons in a cardigan.

Set-theoretic binary

Some set-theoretic comparisons are used to comparing sizes and the positions of the pockets. This is explained in section 6.2.2.3.

Symbolic: Likert Matrix

Symbolic features are compared using a matrix of values. By definition, the values in the leading diagonal will be set to the maximum score. The remaining values are provided by the user as Likert scores (see table 6.3), and are symmetric about the leading diagonal. For example, the options for button position might be as shown in table 6.4.

Stitch Patterns

There are three sub-components to the similarity of a stitch pattern:

- The type of pattern (t)
- The stitch tension (h)
- The row tension (v).

The user provides weights to reflect the importance of these, and the similarity is calculated thus:

$$s(p_a, p_b) = \frac{w_t s(t_a, t_b) + w_h s(h_a, h_b) + w_v s(v_a, v_b)}{w_t + w_h + w_v}$$

6.2.2.3 List of features

The global similarity is obtained by summing the local similarity of the features in table 6.5, as described above. The options for the Likert matrix features are given in appendix C.

The size match (3) works by comparing the set of sizes from each of the two garments. If these sets are disjoint then the local similarity score is 0, otherwise it is 1.

Features 26-27 concern the pockets, and these work by comparing the set of options for pocket positions that the two garments possess. The possible options for pocket position are given below (the centre option is only available on sweaters):

- left side
- left front
- centre
- right front
- right side.

Let $\mathsf{P_Q}$ be the set of options for the query case, and $\mathsf{P_R}$ for the retrieved case.

- A garment is more similar to another if they have the same number of pockets. If $\mathsf{P_Q}$ and $\mathsf{P_R}$ have the same cardinality then the score for feature 26 is one, otherwise it is zero.
- A garment is more similar to another if the pockets are in the same position. If $\mathsf{P_Q}$ is a subset[4] of $\mathsf{P_R}$and $\mathsf{P_Q}$ is not an empty set, then the score for feature 27 is one, otherwise it is zero.
- A garment is more similar to another if it has no superfluous pockets. If $\mathsf{P_R}$ is a subset[4] of $\mathsf{P_Q}$ then the score for feature 28 is one, otherwise it is zero.

[4] it is not necessary to be a *proper* subset

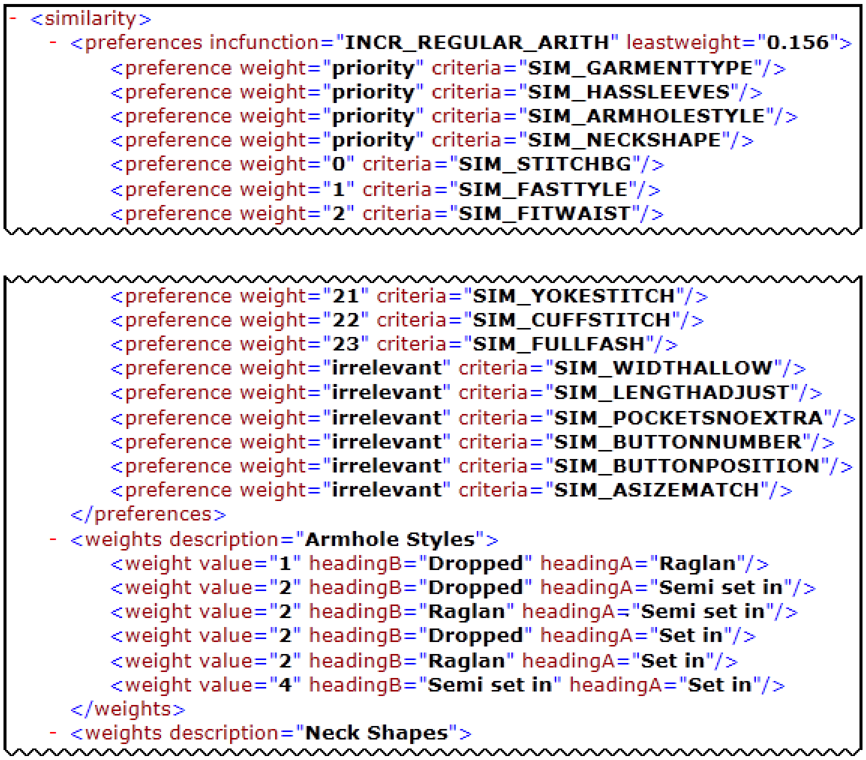
Ref	*Category*	*Description*	*Type*	*Relevance*
1	general	Type of background stitch	SP	universal
2		The option for wearer	LM	universal
3		Whether or not at least one of the sizes matches	STB	universal
4		Length adjustment	RN	universal
5		Width allowance	RN	universal
6		Armhole style	LM	universal
7	body	Shape of neck	LM	universal
8		Waist fitting option	LM	universal
9		Whether there is a bottom border or not	LB	universal
10		The stitch pattern used on the bottom border	SP	if Q has a bottom border
11		Whether or not there is a yoke	LB	universal
12		The stitch pattern used on the yoke	SP	if Q has a yoke
13		The type of garment (cardigan or sweater)	E	universal
14		Whether there is a front border or not	LB	if Q is a cardigan
15		The stitch pattern used on the front border	SP	if Q is a cardigan with a front border
16		Fastener type	LM	if Q is a cardigan
17		Number of buttons	E	if Q is a cardigan with buttons
18		Button position	LM	if Q is a cardigan with buttons
19	sleeves	Has sleeves, or is sleeveless?	LB	universal
20		Straight part at the top of the sleeve	LB	if Q has sleeves
21		Straight part at the bottom of the sleeve	LB	if Q has sleeves
22		Whether it is fully fashioned or not	LB	if Q has sleeves
23		Sleeve length	LM	if Q has sleeves
24		The option for cuffs	LM	if Q has sleeves
25		The stitch pattern used on the cuffs	SP	if Q has sleeves and cuffs
26	pockets	How many pockets there are	E	universal
27		Position of the pockets	STB	if Q has at least one pocket
28		Whether there are no superfluous pockets	STB	universal
29	collar, hood or neck band	Choice of neck option	LM	universal
30		The stitch pattern used on the collar	SP	if Q has a collar
31		The stitch pattern used on the hood	SP	if Q has a hood
32		The stitch pattern used on the band	SP	if Q has a neck band
33		Style of collar	E	if Q has a collar
34		Style of hood	E	if Q has a hood

(a) List of features

E	Equality
LB	Logical Biconditional
LM	Likert Matrix
RN	Real Number
SP	Stitch Pattern
STB	Set-Theoretic Binary

(b) Legend for the types

Table 6.5: Features that are compared



```
- <similarity>
  - <preferences incfunction="INCR_REGULAR_ARITH" leastweight="0.156">
      <preference weight="priority" criteria="SIM_GARMENTTYPE"/>
      <preference weight="priority" criteria="SIM_HASSLEEVES"/>
      <preference weight="priority" criteria="SIM_ARMHOLESTYLE"/>
      <preference weight="priority" criteria="SIM_NECKSHAPE"/>
      <preference weight="0" criteria="SIM_STITCHBG"/>
      <preference weight="1" criteria="SIM_FASTTYLE"/>
      <preference weight="2" criteria="SIM_FITWAIST"/>
```

```
      <preference weight="21" criteria="SIM_YOKESTITCH"/>
      <preference weight="22" criteria="SIM_CUFFSTITCH"/>
      <preference weight="23" criteria="SIM_FULLFASH"/>
      <preference weight="irrelevant" criteria="SIM_WIDTHALLOW"/>
      <preference weight="irrelevant" criteria="SIM_LENGTHADJUST"/>
      <preference weight="irrelevant" criteria="SIM_POCKETSNOEXTRA"/>
      <preference weight="irrelevant" criteria="SIM_BUTTONNUMBER"/>
      <preference weight="irrelevant" criteria="SIM_BUTTONPOSITION"/>
      <preference weight="irrelevant" criteria="SIM_ASIZEMATCH"/>
    </preferences>
  - <weights description="Armhole Styles">
      <weight value="1" headingB="Dropped" headingA="Raglan"/>
      <weight value="2" headingB="Dropped" headingA="Semi set in"/>
      <weight value="2" headingB="Raglan" headingA="Semi set in"/>
      <weight value="2" headingB="Dropped" headingA="Set in"/>
      <weight value="2" headingB="Raglan" headingA="Set in"/>
      <weight value="4" headingB="Semi set in" headingA="Set in"/>
    </weights>
  - <weights description="Neck Shapes">
```

Figure 6.4: Sample portions of the similarity file

6.2.3 Implementation Issues

6.2.3.1 Persistence

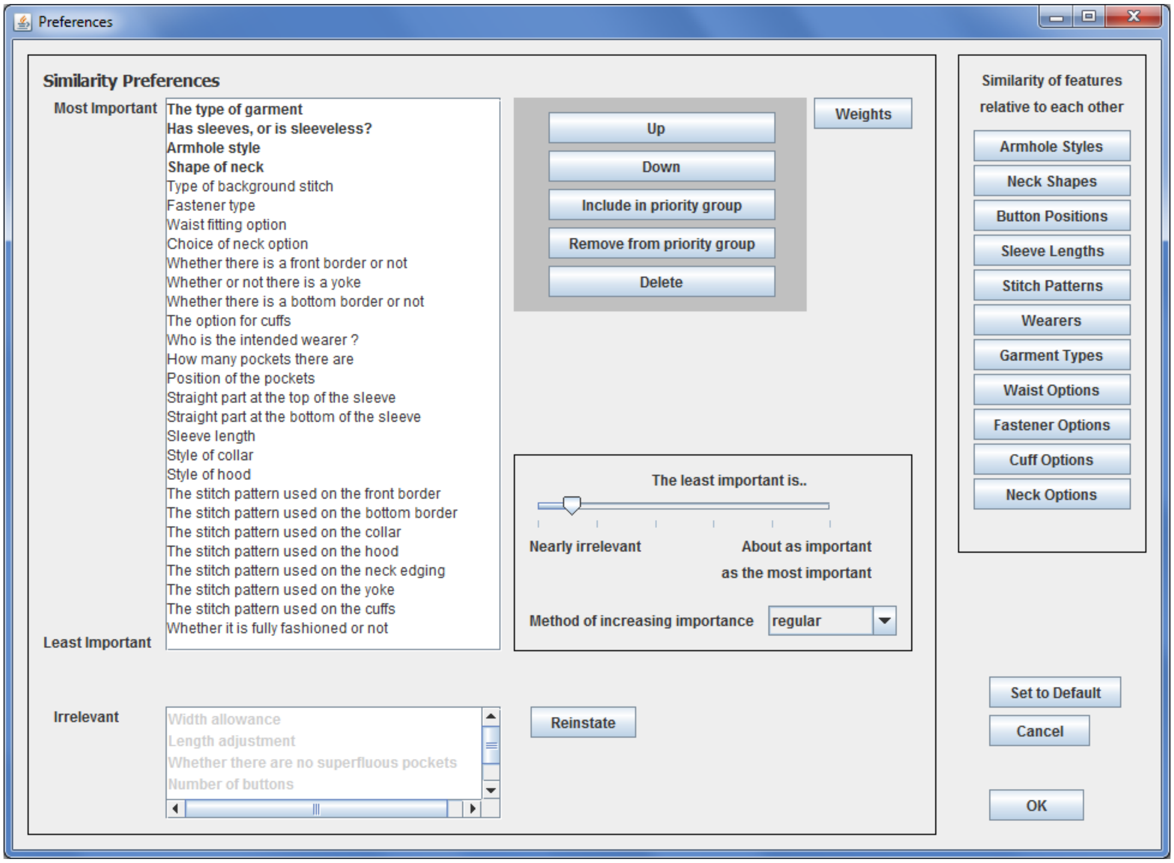
SEACOP stores the similarity preferences in volatile memory for fast access by the garment comparison algorithm. The preferences consist of:

- The lowest non-zero weight
- The function used to generate the remaining weights
- The composition of the priority group
- The composition of the irrelevant group
- The ranks of the ranked group
- The scores for Likert matrices. Due to symmetry and identity, values are only stored where $i > j$.

For persistent storage, the preferences (as listed above) are kept in an XML file. Sample portions of such a file are shown in figure 6.4.

6.2.3.2 User Interface

SEACOP includes a GUI-based editor so that the user can change the preferences [14, 97], as shown in figure 6.5.


Preferences

Similarity Preferences

Most Important

- The type of garment
- Has sleeves, or is sleeveless?
- Armhole style
- Shape of neck
- Type of background stitch
- Fastener type
- Waist fitting option
- Choice of neck option
- Whether there is a front border or not
- Whether or not there is a yoke
- Whether there is a bottom border or not
- The option for cuffs
- Who is the intended wearer ?
- How many pockets there are
- Position of the pockets
- Straight part at the top of the sleeve
- Straight part at the bottom of the sleeve
- Sleeve length
- Style of collar
- Style of hood
- The stitch pattern used on the front border
- The stitch pattern used on the bottom border
- The stitch pattern used on the collar
- The stitch pattern used on the hood
- The stitch pattern used on the neck edging
- The stitch pattern used on the yoke
- The stitch pattern used on the cuffs
- Whether it is fully fashioned or not

Least Important

Irrelevant

- Width allowance
- Length adjustment
- Whether there are no superfluous pockets
- Number of buttons

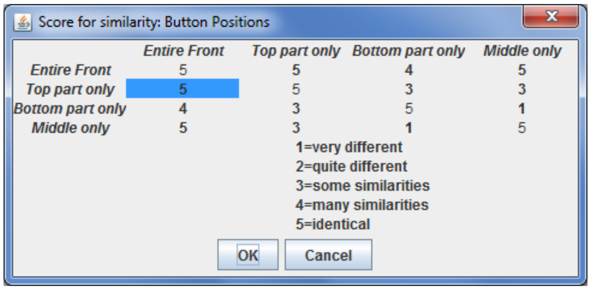
The least important is..

Nearly irrelevant — About as important as the most important

Method of increasing importance: regular

Similarity of features relative to each other

Figure 6.5: User interface for setting similarity preferences


Score for similarity: Button Positions

	Entire Front	*Top part only*	*Bottom part only*	*Middle only*
Entire Front	5	5	4	5
Top part only	5	5	3	3
Bottom part only	4	3	5	1
Middle only	5	3	1	5

1=very different
2=quite different
3=some similarities
4=many similarities
5=identical

Figure 6.6: Likert matrix example

Each button in the column towards the right of the window corresponds to a Likert matrix. If the user presses a button then a new window appears, allowing the user to set the Likert scores, as shown in figure 6.6.

6.3 Sketch Similarity

6.3.1 Sketch similarity contrasted with related problems

A sketch in SEACOP is implemented as a list of so-called Shape Point objects; these are points in the 2-D Euclidean plane, which contain additional contextual information as described in section 5.2.5.1. Most Shape Points are connected to the adjacent Shape Points in the list by a line segment, Bézier curve or elliptical quadrant.[5] Sketch comparison is different from the comparison in many problems discussed in the literature, in the following ways:

- *It is not image comparison*: the important points are already defined by the Shape Points, so keypoint extraction [160] is not necessary. For our purposes there is no colour, shading or texture and therefore histogram-based techniques are not relevant [161].
- *It is not a 3-dimensional problem* such as the analysis of point clouds [162].
- *The only transformation which is invariant is translation*. Therefore, any techniques which ignore rotation or scaling [163] are not applicable.
- *Sketches are not graphs*; although they share some characteristics of a labelled graph, the location of the Shape Points (vertices) is relevant. Therefore it is not a graph matching or editing problem [164].
- *Sketches are not polygons*; they often have connectors which are not lines. Also, to use a graph-theoretic metaphor, they may have more than one cycle (e.g. if there is a yoke). Therefore techniques which assume they are are polygons [165] are not applicable. Even if two sketches were polygons, they would not necessarily have the same number of line segments ("sides"). It is also worth noting that the sketches can contain concave regions.

Additionally, sketches are heterogeneous; they do not necessarily contain exactly the same features. However, sketch comparison is not the same problem as comparing random shapes; in general each type of piece has specific features. A sleeve, for example, will be distinguishable from a front.

6.3.2 Measures based on translation

The first stage in determining sketch similarity is mapping; the details of this are described in section 7.2.2. This process is invoked for each type of piece that the two garments have in common; for each such piece it outputs a set of pairs of Shape Point objects. Within each pair, the two Shape Points will be from different garments. The measure of sketch similarity is based on the distances between the Shape Points in each pair.

6.3.2.1 Translation issues

A naive measure of similarity might average the distances between pairs of points, however this approach ignores issues of translation. It is unlikely that a user would translate a whole piece, since that would

[5] There are other Shape Points that are not connected in this way, which are used for buttons or for other implementation purposes.

mean moving each individual Shape Point.[6] However, when editing a piece, there is often more than one way to achieve the same goal. For example, shortening a sleeve can be done by editing either end; the cuff can be moved towards the crown, or vice versa. These two methods would result in the same shape, but several Shape Points would have locations that differ from one version of the sketch to the other.

One option for dealing with translation issues is to apply one translation to each of the pieces in a garment, so that they map as closely as possible on to the equivalent piece in the other garment. The sketch would not actually be changed, but the translation would be taken into account when calculating the distances between the Shape Points in the pairs; the distance that is used in the calculation is the one between a translated version of one of the Shape Points, and the other Shape Point.

The issue then is how such a translation is determined. Each piece has one of its Shape Points designated as the origin, so it is possible to translate one piece such that the origins have the same coordinate. However, the user can move the origin; its location is not necessarily at (0,0) and does not hold any special significance (other than the fact that it defines the axis of reflection).

6.3.2.2 Bounding box distance (BBD)

The *bounding box distance (BBD)* process is defined as follows. BBD scales one of the pieces, so that its bounding box has the same dimensions as the other piece. The translation that is applied is the one required to align the bounding boxes. The translation is ignored, so the output of the BBD process is the average distance between the points in the pairs and the scaling.

A simple test rig was constructed (outside of SEACOP) to test this algorithm. One of the shapes used is shown in figure 6.7. The points are numbered A to J and the integers refer to the distances between these points, e.g. I is 17cm to the right of G. The tests subjected the shape to an operation which involves translations, and then compared the resulting shape with the original. In each of the operations, the same translation was used for each of the points that were affected. The operations were:

1 *Move border down:* The y-coordinate of points C and D is translated.

2 *Move bottom down:* The y-coordinate of points A and B is translated.

3 *Move both down:* The y-coordinate of points A, B, C and D are translated.

The results of the testing are shown in table 6.6. Each column relates to a progressively larger translation. The rows give the results for BBD (both scale and distance) compared to another distance measure which is referred to here as the Minimum Average Distance (MAD is explained in section 6.3.2.3).

The results show that MAD increases with the magnitude of the translation. Also, the MAD for operations 1 and 2 are the same, but that of operation 3 is double. These results are intuitively what is

[6] The exception is a cardigan front; the user interface provides an easy means to translate this laterally (see section 5.2.4.4).

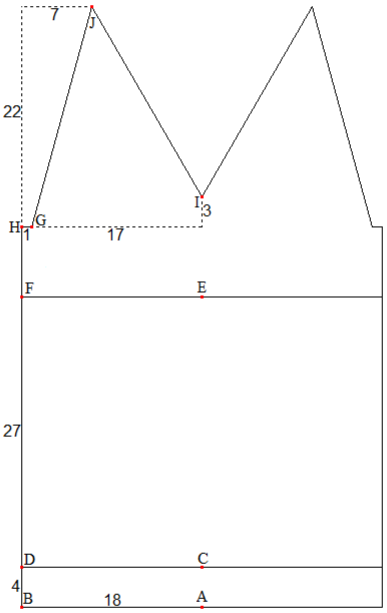


Figure 6.7: Example shape

Operation	*Datum*	*Parameter Value*					
		0.5	*1*	*1.5*	*2*	*2.5*	*3*
1. Move border down	MAD distance	0.0938	0.1875	0.2812	0.375	0.4688	0.5625
	BBD scale-X	1	1	1	1	1	1
	BBD scale-Y	1	1	1	1	1	1
	BBD distance	0.1	0.2	0.3	0.4	0.5	0.6
2. Move bottom down	MAD distance	0.0938	0.1875	0.2812	0.375	0.4688	0.5625
	BBD scale-X	1	1	1	1	1	1
	BBD scale-Y	1.0083	1.0167	1.025	1.0333	1.0417	1.05
	BBD distance	0.1942	0.3883	0.5825	0.7767	0.9708	1.165
3. Move both down	MAD distance	0.1875	0.375	0.5625	0.75	0.9375	1.125
	BBD scale-X	1	1	1	1	1	1
	BBD scale-Y	1.0083	1.0167	1.025	1.0333	1.0417	1.05
	BBD distance	0.1075	0.215	0.3225	0.43	0.5375	0.645

Table 6.6: Effect of changing the bottom of the example shape

expected; the distance is increasing with the amount of change.

The BBD results also show a distance that increases with the magnitude of the translation. However, in contrast to MAD, the distance for operation 3 is about the same as that of operation 1, and it is that of operation 2 which is approximately double. The reason for the low distance in operation 3 (compared to MAD) is that points A and B lie on the bounding box, therefore only points C and D are included in the distance.

Operation 2 has higher BBD distances than operation 3. Both operations involve the same scaling. However, in operation 3, the translations do not affect the width of the bottom border. The scaling has only a small effect on the border width in operation 3 and hence the distance mainly comes from points E to J. In contrast, operation 3 affects the border width and this results in points C and D also making

Algorithm 6.1 Determination of MAD

```
procedure MAD(input List of ordered pairs of ShapePoints)
        translation := (0,0)
        step := 1
        best := AverageDistance(list, translation)
        while(step>0.001)
                clear changes flag
                for each neighbour of the translation
                        candidate := AverageDistance(list, neighbour)
                        if(candidate<best) then
                                best := candidate
                                translation := neighbour
                                set changes flag
                        end if
                end for
                if changes flag not set then
                        step := step / 2
                end if
        end while
        output best
end
```

a substantial contribution to the overall distance.

The results indicate that (in contrast to MAD) the distances in BBD cannot be used in isolation, but only within the context of the scaling. Although the scaling can be combined into a single value[7], the obvious disadvantage of the BBD approach is that it yields at least two outputs *for each piece*. It is likely that the scalings will be different for each piece, and it is not clear how these can be combined into one value (for a garment) or used otherwise.

6.3.2.3 Minimum average distance (MAD)

The approach that was taken to determining sketch similarity was to use a translation as discussed above; the translation is obtained using a simple search, as per algorithm 6.1. The algorithm takes as its input a list of ordered pairs of Shape Points, these are the output of the mapping process. In each pair, the first Shape Point is from the garment to be kept fixed, the second Shape Point is from the garment containing the pieces that will be translated. The output of the algorithm will be the *minimum average distance* (MAD).

Algorithm 6.1 begins with the identity transformation, and calculates the average distance according to algorithm 6.2. Then, the algorithm iterates through a procedure which is similar to a binary search. At each iteration, it compares the best translation so far with all its neighbours. If the best translation is (x, y) then the neighbours will be the set $\{(x-s, y-s)$, $(x, y-s)$, $(x+s, y-s)$, $(x-s, y)$, $(x+$

[7] By taking the square root of the sum of the squares of the horizontal and vertical scale factors.

Algorithm 6.2 Determination of average distance

```
procedure AverageDistance(inputs: List of ordered pairs of ShapePoints,
                                                          translation)
        totalDistance := 0
        for each pair in the list
                pA := the first ShapePoint in the pair
                pB := the second ShapePoint in the pair
                pB' := pB which has been subjected to the translation
                d := distance from pA to pB'
                totalDistance := totalDistance + d
        end for
        output totalDistance/(size of the list)
end
```

$s, y)$, $(x+s, y+s)$, $(x-s, y+s)$, $(x+s, y+s)\}$ where s is the step value. The step value s is initialised to 1cm. The translation starts as (0,0) but if any of the neighbours proves to be better than the (previously found) best translation, the record is updated accordingly.[8] A better translation is defined as one with a lower average distance (as determined by algorithm 6.2). After all the neighbours have been examined, if the best translation was not updated then the step value is halved. Algorithm 6.1 terminates when the step value falls below a preset (small) value.

Algorithm 6.1 can be visualised thus: if two similar shapes are drawn on two transparent sheets, and those sheets are overlaid on each other, then one can be manually manipulated until the two shapes are in as similar position to each other as possible. A simple example is shown in figure 6.8a; there are two shapes which resemble sketches of the front of a v-neck sweater. The shape shown in the dashed outline is shorter and wider than the one presented with a solid line; it also has a shallower neck and less of an arm inset than the other shape. Figure 6.8b shows the results of applying the translation used to calculate MAD.

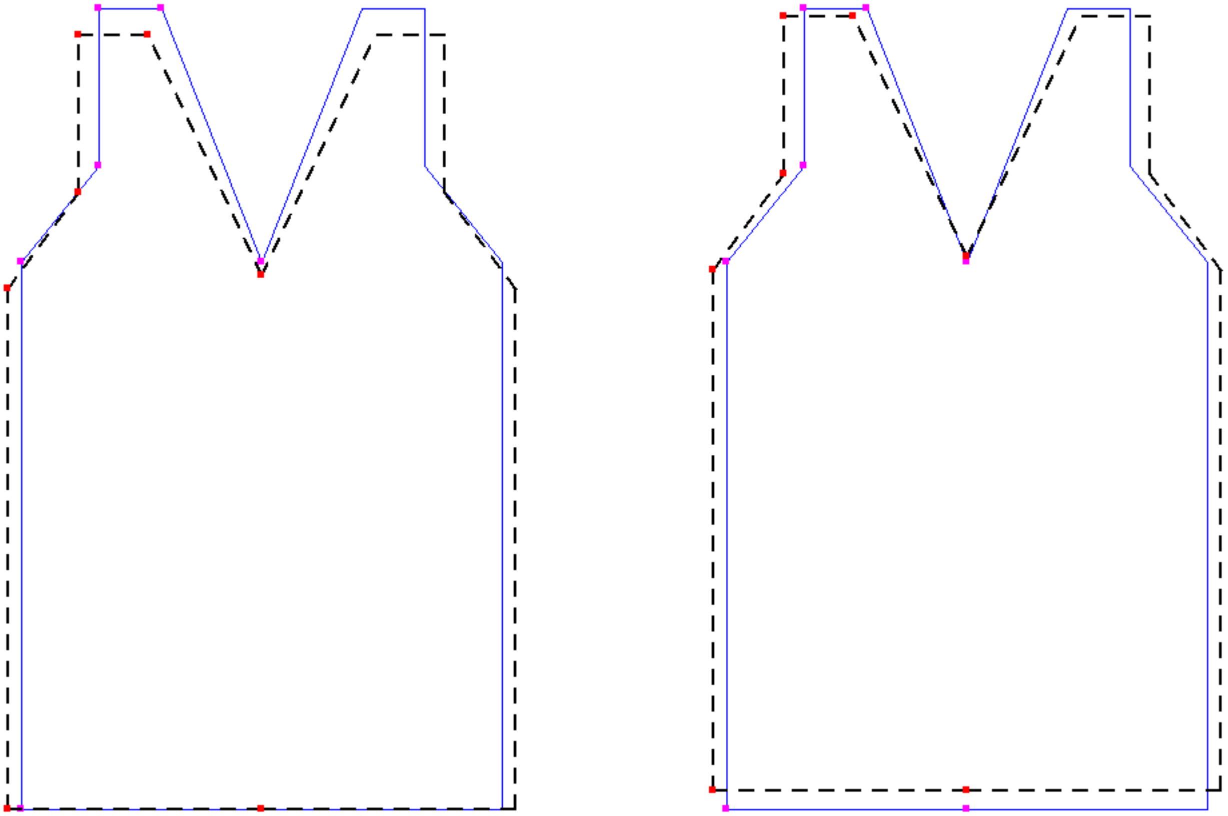
Since the pieces are all subjected to the same translation, the result can be aggregated for the garment as a whole. A key advantage of MAD is that there is only one numerical output (the translation values are ignored after they have been used to calculate MAD). This contrasts with the bounding box algorithm in section 6.3.2.2.

6.3.2.4 MAD and local minima

MAD was defined in section 6.3.2.3, and an example illustrated in section 6.3.2.2 where it seemed to give intuitive results. However, algorithm 6.1 assumes that there is only one local minimum. So, a simple experiment was required to provide evidence for (or against) this assumption.

The experiment took the form of a Monte-Carlo simulation [166]. For each run of the experiment, the

[8] Updating the record of the best translation whilst the iteration of the neighbours has not been completed does not affect the locations of the remaining neighbour points.



(a) Before

(b) After

Figure 6.8: Simple example of a translation used to implement MAD

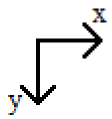


Figure 6.9: Coordinate system

#	*Operation*	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*
i	Lengthen	y+	y+	y+	y+						
ii	Widen		x-*		x-*		x-*	x-*	x-*		x-*
iii	Thicken bottom border			y-	y-						
iv	Move yoke down					y+	y+				
v	Decrease armhole depth							y-	y-		
vi	Decrease armhole inset										x-*
vii	Decrease neck depth									y-	

Table 6.7: Operations

shape in figure 6.7 was subjected to a change. The MAD between the original and the changed version was then calculated. The experiment was conducted using a test rig, as per the experiment in section 6.3.2.2. The following conventions were adopted:

- The unit is centimetres.
- The coordinate system is as shown in figure 6.9.
- Point A is the origin.

The changes to the shape that were made using the operations are listed in table 6.7; each one takes a single numeric parameter which affects a coordinate of one or more points. The code refers first to the coordinate which is changed (x or y), then to the effect on that coordinate (- or +). If the code is suffixed by an asterisk (*) then the coordinate is affected by only half the parameter value.

The operations are subject to constraints as shown in table 6.8; each constraint takes the form of a permissible range for its parameter. The shapes also have constraints, as shown in table 6.9; each constraint affects only two points.

In each run of the simulation, a copy of the shape was made, which was then subjected to changes.

#	*Operation*	*minimum*	*maximum*
i	Lengthen	-6	6
ii	Widen	-6	6
iii	Thicken bottom border	-3	6
iv	Move yoke down	-6	6
v	Decrease armhole depth	-6	6
vi	Decrease armhole inset	-2	2
vii	Decrease neck depth	-6	6

Table 6.8: Constraints on operations

#	Constraints	Explanation
1	B.x < A.x	body cannot have negative thickness
2	C.y < A.y	bottom border cannot have negative thickness
3	E.y < C.y	yoke must be above the bottom border
4	H.y < F.y	yoke must be below the armhole
5	I.y < E.y	yoke must be below the neckline
9	J.y < I.y	neck must have a depth
11	J.x < I.x	neck must have a non-negative width

Table 6.9: Constraints on shapes

Algorithm 6.3 Random MAD Algorithm

```
Procedure Random-Search(shapeA, shapeB)
        best-translation := (0,0)
        result :=0

        for 100 iterations
                translation := (randomly chosen x, randomly chosen y)
                run MAD with (shapeA, shapeB, translation)
                if the output of MAD is better than result then
                        best-translation := translation from MAD output
                        result := result of MAD output
                end for

        output best-translation and result
End
```

Each of the operations in table 6.7 had a 50:50 probability of being selected, thus there was usually more than one change. Then, the constraints in table 6.9 were checked. If any were violated, the changed shape was discarded and the random changes process was then repeated with a new shape, as necessary, until a valid (changed) shape was obtained.

The MAD between the valid shape and the changed shape, using algorithm 6.1, was recorded. The output of the Random MAD Algorithm (algorithm 6.3) was also recorded; this is a form of random search which invokes algorithm 6.1 repeatedly. When algorithm 6.1 is ran in isolation, a null translation is normally passed as a parameter to form the starting point of the search. In contrast, algorithm 6.3 uses a random translation on each search, and returns the best result.

The result of algorithms 6.1 and 6.3 were compared, for 10000 successful iterations of the simulation. The maximum absolute difference between the two values was 7 x 10^{-7} cm (to 1 significant figure). In 76.48% of results, the two outputs were identical, to within the accuracy allowed by a Java double-precision decimal. The differences between the outputs of the two algorithms are so small that they lead to the conclusion that it is likely that there is only one global minimum for the problem that was investigated in this experiment. Small differences may arise simply due to rounding errors.

The experiment was then repeated for three other shapes (which are not shown). The shape in figure

6.7 was intended to approximate to the front of a Raglan garment. The other shapes were:

- Round-neck front with a set-in shoulder and a yoke
- Round neck front with a dropped shoulder
- Round neck garment with a set-in shoulder and a fitted waist.

Similar results were obtained with the other three shapes: the maximum deviation was of the order of 10^{-7} cm. From this it can be concluded that the random search in algorithm 6.3 is unnecessary. Algorithm 6.1 is likely to return a value that represents one global minimum, and is therefore used where MAD is referred to hereafter.

6.3.3 Proportion of Common Shape Points (PCSP)

One of the drawbacks of algorithm 6.1 (MAD) is that it only takes into account the Shape Points that are in common to the two pieces. If Shape Points are present in one garment but not another, then presumably this makes them different, however that is not taken into consideration when MAD is calculated. For example, if a copy was made of a back of a sweater, and a yoke added to the sweater, then MAD between the back and its copy would be zero, indicating that the two garments are identical (which is obviously wrong).[9]

One way around this is to include Shape Points in the mapping where there is no equivalent in the other garment. In lieu of an actual distance for the pair, the calculation could use a penalty value instead. The drawback of this approach is that the choice of penalty value is arbitrary.

Another approach is to develop an additional measure of sketch similarity, the *Proportion of Common Shape Points* (PCSP). This is given by:

$$p = \frac{m}{m + u_1 + u_2}$$

Where:

- m is the number of mapped Shape Points
- u_1 is the number of unmapped Shape Points from garment (1)
- u_2 is the number of unmapped Shape Points from garment (2).

The primary reason for computing sketch similarity is as a measure of solution similarity, and in PCSP presented a number of implementation difficulties in this regard. Firstly, assume that the conditions on

[9] Although, as explained in chapter 7 and the final paragraphs of this section, this situation will not arise in adaptation. If the user specifies that a yoke is required, the result of adaptation will include a yoke; however there is no guarantee that the yoke will be exactly in the location that the user required.

mapping for MAD (as described in section 7.2.2.2) are used with the PCSP algorithm. Then, if the output of adaptation (as explained in chapter 7) is compared with the ideal output, the result of PCSP would always be 1. This is because SEACOP's adaptation (as explained in chapter 7) does not add user added points, bands or panels, and these types of Shape Points are excluded from the similarity mapping by the conditions in section 7.2.2.2. When using the adaptation algorithms described in chapter 7, the questionnaires of the output of adaptation and the ideal result are the same. The questionnaire composition determines which Shape Points are part of the garment. Hence, every Shape Point that is eligible to be mapped is in fact mapped, giving a result of 1.

An alternative to applying the conditions is to map every Shape Point between the two sketches. This becomes problematic if any two sketches can be used, because it means that user added points, bands and panels are included. How can the Shape Points in panels in two different garments be mapped to each other? Panels have much less well defined roles than the Shape Points that are created by the sketch instantiation algorithm. For example, the user is free to add extra Shape Points to the boundary of a rectangular panel, and to translate, reflect and rotate it. In general, it is difficult to devise an algorithm that determines if a Shape Points on panels are equivalent to each other.

Since sketch comparison is used as a means of comparing solutions, another option would be to include panels, bands and user-added points, but to stipulate that only one of the two garments may have these. This then introduces a complication that the PCSP algorithm cannot necessarily be used to compare a garment with itself; thus, we cannot verify if it meets the identity constraint for a metric. As chapter 7 explains, the compositions of the output of adaptation and the goal of adaptation are qualitatively very similar. The only difference between the two is the presence of panels, bands and user-added points; these may be present in the goal but will not be present in the output. A PCSP calculation with all types of Shape Points included would simply measure the presence of these specialist features, and thus is likely to be of restricted benefit. Therefore, PCSP was not adopted as a measure of sketch similarity.

Chapter 7

Adaptation

The phrase "adaptation is the Achilles heel of CBR" is so common in research [110, 167, 168, 169] that it has become a cliché. However, it is true that adaptation is the most difficult stage of CBR. This chapter describes how adaptation was designed and then performed in SEACOP. Section 2.1.1 describes the factors which make adaptation particularly complex in the knitwear domain, and introduces a strategy (called *rule difference replay*) which is designed to cope with that complexity. Finally, in section 7.2, the specific way in which the strategy is implemented in SEACOP is explained.

7.1 Introduction

Section 7.1.1 describes the factors which make adaptation particularly complex in this domain. Adaptation in SEACOP is a challenging problem because of the size of the search space, the constraints that apply to the cases, and the heterogeneous nature of those cases. Section 7.1.2 explains why null adapation is not applicable. Then, section 7.1.3 introduces a new type of adaptation operator called *rule difference replay*.

7.1.1 Knitwear design specific aspects of adaptation

The size of the case base is relevant since in reality it will always be small in comparison to the problem space, as defined by the number of questionnaire options. Similarity for retrieval is based on comparing questionnaires, and there are at least 10^{15} possible combinations of questionnaire options. Not all of these options are likely to occur, but even if a small proportion are used the search space is still very large, and is likely to be several orders of magnitude larger than the size of any case base. This is highly relevant since in CBR, similarity is assumed to correlate with adaptability [4]. If the coverage of the case base is sparse and so the nearest neighbour is very distant, then adaptation will presumably be more difficult.

The various *constraints* in knitting are discussed in section 4.3. The changes instituted by adaptation may violate those constraints, and then it could be difficult to 'repair' those violations without simply undoing the changes. For example, if the query case was a sleeved garment and a sleeveless one is retrieved, reuse of the armhole may violate sleeve-armhole consistency.

Heterogeneity is a notable feature of the questionnaires in SEACOP. Some questionnaire options affect which values are permissible for other options, for example a cardigan cannot have a centre pocket. Other options determine which features are present in the sketch. For example, the location of the ends of the front border will be different depending on whether a neck band or bottom border are present. If the retrieved and query cases are composed of different features, reuse may be not applicable.

Adaptation operators were discussed in section 2.2.3.1. It would presumably be possible to accomplish adaptation in SEACOP using a combination of substitution and transformation. For example, if the query case specifies a pocket but the retrieved case lacks one, then the pocket could be added (substitution) by an adaptation rule. However, the locations of the pocket(s) might depend on the option for the waist: a fitted waist is tapered, and hence if the pocket is intended to be at the edge of the garment, this will affect its position. Also, the positions of buttons could be reused, but by convention the location of buttons is affected by the gender of the wearer[1]. The retrieved case and query case might have a different waist option, or a different gender.

It would be necessary to build the knowledge required to cope with heterogeneity into the adaptation mechanism. The resulting knowledge engineering and software engineering process would likely be lengthy, complex and risky. The large number of combinations of scenarios would make the resulting program difficult to test, and there would be a high risk of error due to incompatibilities between the set of features in the query and retrieved cases. SEACOP's sketch instantiation algorithm (described briefly in section 5.2.7.1) is an illustration of such complexity; it took many person-months to implement. It consists of over 1600 lines of complex program code. The complexities of dealing with the combinations in one case (such as in the sketch instantiation algorithm) are difficult, and those of dealing with two are likely to be harder still.

Knowledge intensive CBR systems arguably negate the main advantage of CBR: the elimination of the knowledge elicitation bottleneck. Often, the reason why CBR is used is because rules are difficult to formulate; if so, these difficulties will hinder the development of a knowledge-intensive system. In a domain such as fashion, flexibility is important and a important advantage of CBR is that it can respond dynamically to fashion trends. However, the knowledge contained in the adaptation rules of a knowledge-intensive system is typically fixed, as per the structure created by the knowledge engineer. The difficulty in formulating adaptation rules and their lack of flexibility were the reasons why a knowledge-intensive approach was not implemented in SEACOP. Also, knowledge-intensive CBR systems have been around for about 25 years (e.g. JULIA [170]), and so it is questionable whether the construction of a new one would add scientific value to this thesis.

[1] A woman's buttons are on the left side of a cardigan, but the man's are on the right

Case merging (as explained in section 2.2.3.3) was considered as a possible means of implementing adaptation. Its main advantage to the problem discussed in this thesis is that the individual parts, by definition, have less features than the case as a whole; therefore the search space becomes smaller. Knitwear is naturally partonomic; it is conceivable that individual pieces could be retrieved, then assembled into one garment afterwards. However, the problem of incompatibility between the parts is a serious disadvantage of case merging. Also, the issues highlighted above about substitution and transformation would also apply to the individual pieces. It is conceivable that the *rule difference replay* strategy introduced in this chapter could be implemented with case merging. However, the decision was taken to investigate the efficacy of the strategy with retrieval of whole cases first. The implementation of case merging is left for future work, as discussed in section 9.3.

7.1.2 Null adaptation is not applicable

The option of null adaptation is chosen in many CBR systems, however it would be inappropriate in SEACOP. It is unlikely (due to the large search space compared to the size of the case base), that the retrieved and query cases would have the same set of features. In a null adaptation scenario, it would be the responsibility of the designer to change the features (in the retrieved case), via the questionnaire, so that the required ones were present. One of the limitations of SEACOP is that questionnaire changes cause the sketch to be re-instantiated.[2] This would completely nullify the benefit of CBR, since editing the questionnaire of the retrieved case would cancel any edits made to the sketch.

The reason why this limitation exists is because coping with any arbitrary questionnaire change without resetting the sketch would be an onerous software engineering task, involving many different rules for the different scenarios. For example, if the user changes the armhole style, what should SEACOP do if this now causes it to cross over the neck? If the user wants to add a yoke, what happens if it crosses over a pocket? The sketch instantiation algorithm avoids these issues by using heuristics; the elicitation of these heuristics is simplified by the fact that the sketch is always instantiated in a consistently predictable manner. This assumption would not be valid if the user's edits were required to be preserved.

In order to preserve the sketch during questionnaire changes, the software would need to be augmented with rules. For example, if adding a yoke would mean it crosses over a pocket, then the pocket could be translated so that the problem no longer occurs. However, the process of developing and implementing those rules could be a complex one, involving similar issues to those described in section 7.1.1 for knowledge intensive adaptation. In the previous example, what if the translation then caused the pocket to cross over the bottom border, or the edge of a fitted waist? The rules would presumably be imperfect, since it would be difficult to cater for every scenario. Failure of the rules would mean that the user would

[2] There are a few exceptions to this, e.g. the designer can make the garment sleeveless or change the stitch patterns used without resetting the sketch.

be left with either a corrupted sketch, or a sketch which has been reset (this nullifying their edits). For all these reasons, null adaptation would be (paradoxically) a difficult option to implement; also, it is unlikely to be of use.

7.1.3 Rule difference replay (RDR)

Section 7.1.1 explained the difficulties in implementing adaptation in SEACOP with substitution or transformation, and with case merging. At first glance it is also difficult to see how *derivational replay* could be used to adapt garment designs, since the literature consistently refers to it being used in planning [54, 171]. Derivational replay replays the sequence of decisions that were made, but in design the order that edits are made is irrelevant.

If the order in which the edits were made is ignored then we are left with just the nature of those edits themselves. The edits are relevant, but only useful if they can be applied in a new scenario. *Rule difference replay* (RDR) is defined here as an adaptation operator conceived during this project. RDR is inspired by derivational replay, but as the following sections show, the two techniques are distinct.

RDR works as part of hybrid CBR systems; the other component of the system is called the *supporting process*. In order for RDR to be applicable, a number of assumptions must hold; these are explained in section 7.1.3.1. RDR works by invoking the *supporting process* first, and then secondly by replaying the edits which were previously made to the retrieved case. This is explained in more detail in section 7.1.3.2.

7.1.3.1 Criteria for RDR

The criteria for RDR to be applicable are:

- *Case-based design*: RDR is best suited to synthesis tasks. In planning, derivational replay would be the natural choice. In classification tasks, the emphasis is on similarity rather than adaptation, so RDR is unlikely to be required.

- *Rules*: RDR must be supported by a deterministic generative process (the *supporting process*), which is capable of producing a solution with the correct features in it. The *supporting process* could be a rule-based system, or conditional programming such as SEACOP. RDR is unlikely to be used with stochastic generative systems such as evolutionary algorithms.

- *Inferior rules*: RDR is only applicable when the rules tend to produce inferior results. Although there is typically no one unique correct answer to design problems, this criterion means that the output of rules will usually be of lower quality than that produced by an expert designer. If the output of the rules was as good as a human expert, then the rules would be sufficient by themselves, and thus no hybridisation with CBR would be required. Although the *supporting process* will add the correct features to the solution, the assumption is that quantitative changes will often be required to produce a good solution.

- *Possible to improve*: it must be possible to improve the designs from the inferior state produced by the rules, by altering the features that are present. Also, such changes must normally be easier than producing the design from first principles. If the rules output a solution that is a local minimum, and there is a big distance between this and the global minimum, then this assumption may not hold.

- *Supporting process output recoverable*: there must be a way to recover the output of the supporting process; this is explained in the next section.

- *Commonality*: for RDR to work, there needs to be some common features between the artefacts being produced. RDR would not be an appropriate way to produce art, for example, since it is easily possible for two works by the same artist to have nothing in common.[3]

- *Partial changes possible*: if the designs are heterogeneous, then by definition some features will not be common (see the *commonality* assumption). In this situation, it must be possible to change only some of the features of the design, leaving others as per the output of the supporting process. Also, such changes (where applicable) must normally be beneficial.

The role that these assumptions play in the execution of RDR is explained in the next section.

7.1.3.2 How RDR Works

RDR works using a modified version of the classical CBR cycle [1], as depicted in figure 7.1. As with the classical cycle, we begin with the problem being represented as a new case, the query case. However, the key difference with RDR is that the supporting process (see the *rules* assumption) is then invoked on that query case; by the *inferior rules assumption* this output will often be inferior (to that which a human designer would produce). The next step is then for a case to be retrieved, as per the classical CBR cycle.

Then, the changes that were previously made to the *retrieved case* are replayed on the *rules output case* (which was generated by the supporting process). This is possible because the *supporting process output recoverable* assumption makes it possible to gain visibility to the changes that occurred in the retrieved case, from when it was originally itself the output of rules. Due to the *commonality* assumption, there will be common features between the *retrieved case* and the *rules output case*; a mapping can therefore be obtained between the two cases and the way in which the features were changed in the *retrieved case* can be applied to the *rules output case*. Due to the *possible to improve* and *partial*

[3] It is unlikely that CBR in general would be a good way of producing art, so the *commonality* assumption not unique to RDR.

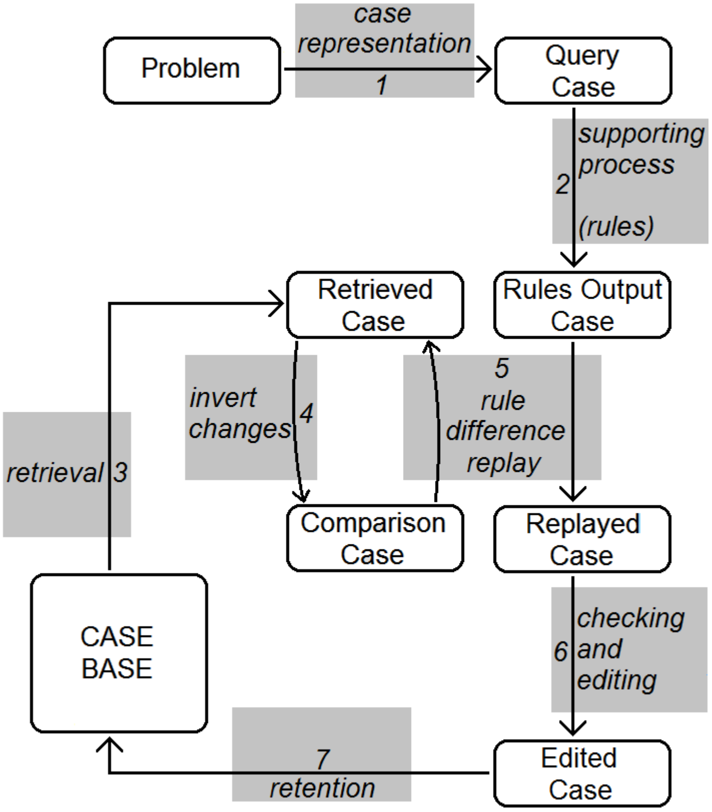


Figure 7.1: CBR Cycle modified by RDR

changes possible assumptions, we know that all or part of the *rules output case* will be editable, and the edits can be carefully chosen to improve the quality of the case.

In figure 7.1, a *comparison case* is shown; one way to implement RDR is to make a copy of the *retrieved case*, and undo the changes that were made to it to form the *comparison case*. Then, the *retrieved case* and *comparison case* are compared. However, RDR could also be implemented without the *comparison case*: it may be sufficient to keep a record of the changes that occurred and to then determine the inverse of the changes in the record.

The result of RDR is known as the *replayed case*. We do not necessarily assume that this is ideal, so the *replayed case* is checked to see if it can be improved; this is step 6 in figure 7.1. If necessary, it is improved and then the *edited case* is retained in the case base for future use. Presumably the methods of editing in step 6 are manual; if an algorithm could be used then it may be possible to combine that algorithm with the supporting process to avoid the use of CBR altogether. The objective of RDR is to reduce the amount of manual editing through the application of CBR.

Step 6 is the only one that involves intervention from the user. It may be possible to reduce the burden of this intervention if an objective assessment of fitness can be formulated; this is the same issue as occurs in evolutionary algorithms. However, the "fitness" of a design is often determined by a variety of factors including aesthetics; these could be impossible to measure objectively.

7.1.3.3 RDR is a method of hybridisation

In hybrid systems in general, the two components of the system normally[4] complement each other and work symbiotically. In RDR, the predictability of the rules is enhanced by the flexibility of CBR. In the sort of problems that are amenable to RDR, rules are unlikely to be used alone as they are not likely to be sophisticated or flexible enough. RDR is beneficial in problems which are difficult to specify completely, or where the specification is often vague. In these situations the knowledge engineering exercise is likely to be frustrating and incomplete.

If the engineer tries to solve the problem using rules alone, there is a danger that the rules will be made inappropriately specific; this is akin to the phenomenon of overfitting in machine learning [172]. This danger is particularly acute in tasks involving a lot of tacit knowledge, as experts often explain things by giving specific examples, rather than general principles.

RDR is particularly applicable to situations where most of the domain knowledge is general and static, but the details are subject to change; the rules can implement the static knowledge and CBR can do the rest.

Another advantage of RDR is that it could be used to alleviate a common problem with CBR: insufficient case base coverage. Where there is insufficient coverage, it is unlikely that a case will be obtained that has the required features; this is the role of the rules. Those features do not have to be arranged or configured perfectly, since this is the role of the CBR.

A further advantage of RDR is that it has the flexibility to assist with the initial 'seeding' of the case base. When the system is new and there are insufficient cases, then the *rules output case* can be edited directly by the user if required. Then, the result of this editing is stored in the case base as the *edited case*, bypassing RDR to start with. As the case base grows, RDR can be be invoked and hopefully the requirement for editing will be reduced.

The classification system for hybrid systems of Prentzas and Hatzilygeroudis [5] was discussed in section 2.3. Systems using RDR are most likely to be classed as having sequential processing, since the CBR is invoked after the rules. The co-processing category refers to systems with interleaved processes; although it is possible for the rules to be applied after CBR retrieval occurs, the two are independent of each other and thus cannot be said to be interleaved.

Prentzas and Hatzilygeroudis also discussed coupling and conditionality. RDR systems are tightly coupled, because the CBR depends crucially on the output of the rules. They may have either a conditional or compulsory sequence. A conditional sequence implies that an assessment is made of the output of the rules, to determine if CBR is invoked or not. If a human is responsible for this assessment, then this risks causing user fatigue. A compulsory sequence means that CBR is invoked automatically; this works on

[4] Unless the "standalone" category applies [172], in which case the two systems are separate and the user chooses which output to utilise. However, standalone is arguably not a form of hybridisation.

the assumption that the changes made by CBR are positive ones, and reduce the requirement for further manual editing. The conditional sequence mode could refer to the way of seeding the case base, which is suggested above.

7.1.3.4 Challenges in RDR

RDR is intended to work only in situations where the assumptions which were outlined in section 7.1.3.1 hold. It is a significant deviation from the classical CBR cycle. Therefore, although the issues encountered in constructing any CBR system apply, there are special considerations for RDR.

As with any CBR system, a good measure of similarity is important. Ideally, the similarity measure should correlate with the proportion of features that are in common between the *retrieved* and *rules output* cases; or possibly it should be weighted towards the presence of the more important features in both cases. If the two cases have few features (or only features of low importance) in common then the rule difference replay will have limited effect; the performance of the system will therefore be close to the use of the rules alone. This highlights an advantage of RDR however: the rules can be used as an insurance for outlier cases, where CBR performs poorly.

Finally, the mapping of common features could be challenging, depending on the type of case representation used, and whether the cases are homogeneous or not. In reality, designs tend to be heterogeneous, unless the problem is a very specific and limited one.[5] The similarity algorithm, the need for feature mapping and the requirements of the *supporting process* should all be borne in mind when the case representation is designed.

7.2 Sketch adaptation in SEACOP

Rule difference replay (RDR) was introduced in section 7.1.3; it was designed with the needs of SEACOP in mind. SEACOP meets the assumptions of RDR since it is a complex design process that can be partially supported by rules.[6] The output of the rules is normally inferior, but the designs can be edited and improved by the users using the functionality described in chapter 5. The remainder of this chapter explain how adaptation is implemented to produce sketches of garments using RDR. Section 7.2.1 explains how the *supporting process output recoverable* assumption is met by SEACOP. Section 7.2.2 then explains how the *commonality* assumption is met via a process which maps the common features between two garments. Finally, since SEACOP meets all the criteria for RDR, section 7.2.3 shows how it is implemented.

[5] If artefacts are homogeneous then the process is more accurately referred to as configuration (rather than design), as per section 2.1.3.2.

[6] The word 'rules' here is used in the sense of conditional programming, rather than a rule-based system which is partitioned into separate components (inference engine, rule base etc.).

7.2.1 Obtaining the comparison case

In SEACOP, the query case consists of a questionnaire, which acts as a statement of the problem. This is then subjected to the sketch instantiation algorithm, which takes the role of the *supporting process*. This algorithm is, as required by the assumptions of RDR, a deterministic process which creates the correct features in the design, resulting in the *rules output case*. The results of the sketch instantiation algorithm are unlikely to be ideal: fashion changes and the designers like to 'tweak' the measurements and shapes a little; this is the job of the CBR part of the process.

Then, the *retrieved case* is located, using the similarity algorithm described in section 6.2. The questionnaire from the *retrieved case* is copied, to form the *comparison case*. The editable size of the *comparison case* is changed to be the same as that of the *query case*, as the sketches need to be of the same garment size for comparisons to be meaningful. Then, the sketch instantiation algorithm is invoked on the *comparison case*, so that it consists of a sketch as well as a questionnaire.

The questionnaire copying is done in order to preserve the integrity of the *retrieved case*; the two distinct cases are required as the *retrieved case* and *comparison case* need to be compared in order for RDR to work. In some ways, obtaining the *comparison case* is a reversal of the normal CAD process; an alternative would be to store the unedited version of the sketch as well as the edited one. However, questionnaires are small data structures and are easy to copy, and the implementation described here avoids storing data unnecessarily, when that data can easily be obtained algorithmically.

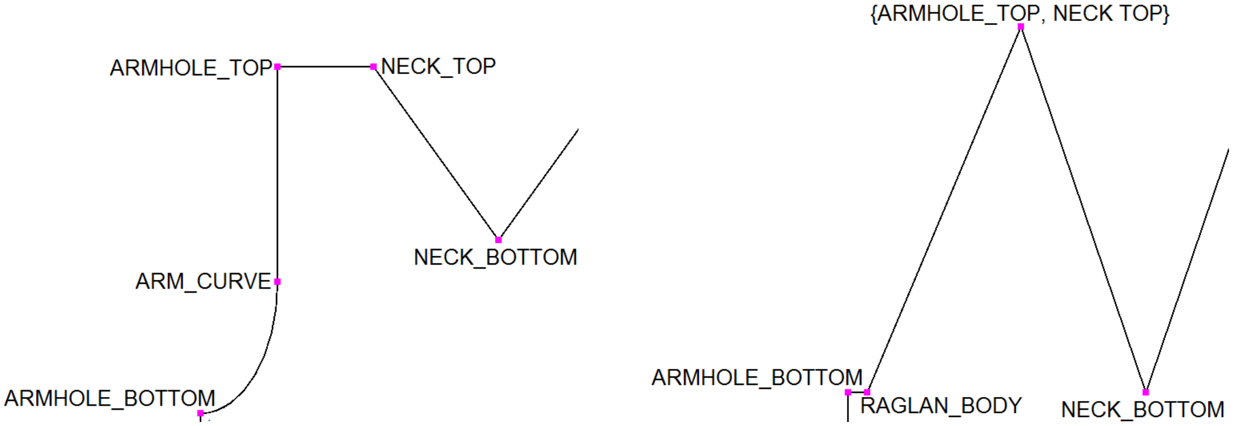
7.2.2 Mapping sketches

The mapping between the Shape Points in the sketches of two different garments is essential for rule difference replay; it is also used to implement the sketch similarity algorithm, which is explained in section 6.3.

The reason why mapping is non-trivial is that sketches are heterogeneous: it is not just that they contain features in different places, they contain different sets of features. This difficulty is overcome, as described below, by labelling Shape Points with *roles*. The output of the mapping process is to establish a list of pairs of Shape Point objects, with each one in the pair from a different garment.

7.2.2.1 Roles

As discussed in section 5.2.5.1, Shape Point objects contain information other than a location; amongst this information is a set of roles. The roles codify the purpose of the Shape Point, for example NECK_BOTTOM occurs at the bottom of the neck. Roles are allocated to Shape Points when they are created by the sketch instantiation algorithm. It is possible for the set to contain more than one role, but currently this only occurs in one situation: a Shape Point in a Raglan body piece has both NECK_TOP



(a) Non-Raglan (b) Raglan

Figure 7.2: Armhole and neck roles in the front

and ARMHOLE_TOP roles; this is illustrated in figure 7.2.

The roles that are present are dependent on the questionnaire options chosen. For example, the roles RAGLAN_TOP and RAGLAN_BOTTOM are only present in Raglan garments. However, some roles are ubiquitous, for example every sleeve will have a Shape Point with the CROWN role (although some garments are sleeveless). There is one important constraint: with some documented exceptions[7], each role will be present only once in a piece.

The roles are used for implementing mapping, for adaptation and sketch similarity (as is explained in the next section). Also, they are used whenever specific Shape Points need to be located for a purpose, for example finding specific points for localised stretches in resizing (see section 5.2.7.2)[8]. Some examples of sketches, annotated with the roles, are shown in appendix D.

7.2.2.2 Criteria for mapping[9]

Roles were explained in section 7.2.2.1. The mapping between two sketches works by comparing the Shape Points in each sketch and pairing together those which have roles in common. However, roles have several purposes (for example, resizing) and so are not necessarily optimised for use in adaptation. Therefore, some additional logic is used during both adaptation and sketch similarity, to ensure that it is appropriate to add a particular pair to the mapping. A mapping is only created if the criteria stipulated in table 7.1 are satisfied. For example, in Raglan garments a point holds the dual role of ARMHOLE_TOP

[7] For example buttons and user-added points all have the same role, so these are not unique; other roles are unique within an element, such as in pockets or panels.

[8] Another example of the use of roles is the retrieval of the Shape Points that are normally located on the left edge of the garment, to determine whether the pocket should be positioned.

[9] In addition to the logic described in this section, some changes are made to the mapping relating to the cuff region of the sleeve, if appropriate. This problem arises because the end of the sleeve has points with the roles CUFF_EDGE and CUFF_MIDDLE in a sleeve with cuffs, whereas in a sleeve without cuffs the corresponding roles are ORIGIN and LEFT_EDGE. If one garment has cuffs and it is being compared to a garment without cuffs, then the mappings are altered to circumvent this problem. In the majority of situations, both garments or neither garments will have cuffs, so this issue will not arise.

ROLE(s)	*Reuse for*		*Comments*
	Similarity	*Adaptation*	
BOTTOM_BORDER_RIGHT	yes	no	In adaptation, only one end is set as we assume move advisers set the other.
YOKE_BORDER_RIGHT			
FRONT_BORDER_TOP			
NECK_BAND_RIGHT			
ARMHOLE_TOP	Yes where there is agreement on whether the armhole style is Raglan or not.		See text.
NECK_TOP			
FITTED_OR_BAGGY_WAIST	Yes where there is agreement on the waist option.		The same role is used for both fitted and baggy waists.
ARM_CURVE	yes	As long as there is agreement on the armhole style.	Semi set-in sleeves have a smaller armhole than set-in but the two are comparable.
ORIGIN	In the front: where there is agreement on whether the garment type is a cardigan.		In a cardigan front, the origin is detached from the garment; this is not comparable to a sweater front.
	Other pieces: yes.		
POCKET_BOTTOM_LEFT	Yes where there is agreement on the position of the pocket.		The centre point functions as a “handle” in the user interface.
POCKET_BOTTOM_RIGHT			
POCKET_CENTRE			
POCKET_TOP_LEFT			
POCKET_TOP_RIGHT			
STANDARD_BUTTON	no	Yes where there is agreement on the number of buttons and the button position.	As button reuse is quite trivial, it is excluded from the similarity assessment.
BAND	no	no	Horizontal and vertical bands are excluded from both symmetry and adaptation.
RECT_PANEL_1	no	no	Rectangular panels are excluded from both symmetry and adaptation.
RECT_PANEL_2			
RECT_PANEL_3			
RECT_PANEL_4			
USER_ADDED	no	no	User added points are excluded from both symmetry and adaptation.
other roles	yes	yes	26 other roles are not specifically mentioned above.

Table 7.1: Criteria for whether mapping occurs

and NECK_TOP (figure 7.2b), whereas in other types of garments the two are separate (figure 7.2a). The two situations are not comparable and so the ARMHOLE_TOP point is not mapped unless either both sketches are Raglan, or neither is.

Some other types of Shape Points are excluded from the scope of sketch similarity and adaptation, and are therefore not mapped. It is difficult to assess what effect panels and bands have on sketch similarity, and they may be difficult to recreate in adaptation due to the heterogeneity of sketches, so these are excluded. Also, adaptation does not create user-added points, as it is difficult to formulate rules about whether adding points is appropriate.

7.2.2.3 Implementing mapping

This section explains how the roles and the criteria discussed previously are used in the execution of the mapping of Shape Points.

Obtaining the set of Shape Points

The mapping process iterates through each piece in a garment, and obtains a set of Shape Points that belong to that piece. The set which is obtained will consist of the Shape Points that are from the part of the piece that is normally editable. The significance of this is that if a symmetric piece is made asymmetric, then only the Shape Points which were editable before the symmetry was removed are relevant because adaptation only creates symmetrical pieces[10]. This set will exclude Shape Points from rectangular panels and bands; as discussed previously these are not relevant to adaptation and they are stored as part of the element (panel or band) itself and not the piece. However, the set will include standard buttons[11] if mapping is being implemented for the purposes of adaptation (but not similarity, since buttons are not involved in similarity as per the previous section).

Eligibility and disambiguation

Once the set of Shape Points has been obtained, the mapping process iterates over each Shape Point and examines its set of roles. The *primary role* is discerned from the set, by the following process:

- In the particular scenario involving Raglan garments described in section 7.2.2.1, the ARMHOLE_TOP role is retained. The NECK_TOP role is discarded, for implementation reasons (the process is simplified by working with only one role, rather than a set). As discussed previously, this is the only situation in which a Shape Point has two roles.

[10] Except in the case of collars or hoods.

[11] But not user-added buttons.

- If the role set is empty, that Shape Point is discarded. Shape Points without roles cannot take part in the mapping process as there is no way to determine their equivalent. In reality, this applies only to a very small number of circumstances such as co-located points.
- If the role set has only one role, then this becomes the *primary role*.

Each pair of Shape Point is then examined to determine *eligibility*, by making reference to and *primary role*. A Shape Point is deemed *eligible* if, in a situation where its parent piece was being mapped to itself, it would pass the rules in section 7.2.2.2. For example, a Shape Point with a primary role of ARM_CURVE will be *eligible* since a garment always has the same armhole style as itself. However a user-added Shape Point will never be *eligible*, since the USER_ADDED role is excluded from mapping without any reference to comparison of the attributes of the two garments.

Mapping

For each type of piece in common to the two garments, a mapping is created. This is done by taking the set of *eligible* Shape Points from each piece, and comparing each element based on their *key*. Only pairs which meet the criteria described in section 7.2.2.2 are incorporated into the mapping. The *key* will be:

- In the case of a standard button, its integer index. The indices are consecutive non-negative integers which are assigned by ordering the buttons by the unique ID of their Shape Points. The Shape Point with the lowest ID is assigned an index of 0, and so on.
- In the case of a pocket, the combination of the pocket position option and the primary role.
- In all other cases, the primary role.

The output of the mapping process is a set of ordered pairs of Shape Points; each item in the pair will have an identical *key* but belong to a different garment. Once the mapping process has been completed, the *keys* can be discarded as they are no longer relevant.

7.2.3 Implementing RDR

This section explains how the concepts introduced previously in this chapter form part of a novel strategy which is implemented for sketch adaptation in SEACOP. As explained in section 7.2.1, adaptation begins with the *query case*, but then the *retrieved case* is obtained and the *rules output case* and *comparison case* are created.

7.2.3.1 Mapping the cases

For adaptation to use RDR, it is necessary for the *comparison case* and *retrieved case* to be compared, and the differences between the two are then applied to the *rules output case*. For this to be effected, there needs to be a mapping between the equivalent Shape Points in all three cases. The mapping between *comparison case* and *rules output case* is created by invoking the process discussed in section 7.2.2. As explained above, this produces a set of pairs of Shape Points; in each pair, one will be from the *comparison case* and one from the *rules output case*.

The mapping between the *retrieved case* and *comparison case* will be more straightforward; these cases contain the same questionnaire, and so will have the same set of features. Each Shape Point has an integer identifier (ID) which is unique within that garment. The sketch instantiation algorithm guarantees that the allocation of IDs will be consistent, i.e. if it is invoked multiple times, each Shape Point will be given the same ID. Therefore, the Shape Points in the *retrieved case* and *comparison case* are mapped simply by comparing their IDs. In fact, the Shape Points from the retrieved case are not necessarily those that are stored in the sketch, since for comparison to be meaningful the garment must be the same size as the query case. So, if the editable sizes of the retrieved and query case differ, then the resizing algorithm (see section 5.2.7.2) is invoked to obtain copies of the retrieved case in the correct size. Since a similar check is carried out for the comparison case (as explained in section 7.2.1), then all three cases relate to garments of the same size.

If there are differences between the *retrieved case* and *comparison case*, this will be because the user has created user-added Shape Points, bands or panels. These things are not features of the sketch instantiation algorithm, but rather are added on demand when the user edits the garment. If present, these features are ignored as they will be found in the *retrieved case* only and so will not be mapped. However this is not a problem since, as table 7.1 specifies, these particular features are outside of the scope of adaptation.

7.2.3.2 Building up the list of proposed moves

As explained above, the mapping process is used to generate a list of pairs of Shape Points, linking the *comparison case* and *rules output case*. We refer to these as p^{C} and p^{S} respectively. The ID-based mapping also provides a link between the *comparison case* and *retrieved case* and we refer to each Shape Point in the latter as p^{R}, and its location as (x_{R}, y_{R}). The Shape Points in the *replayed case* (referred to as p^{P}) are therefore defined thus:

$$x_P = \begin{cases} x_S + x_R - x_C & if\, there\, is\, a\, mapping \\ x_S & otherwise \end{cases}$$

$$y_P = \begin{cases} y_S + y_R - y_C & if\, there\, is\, a\, mapping \\ y_S & otherwise \end{cases}$$

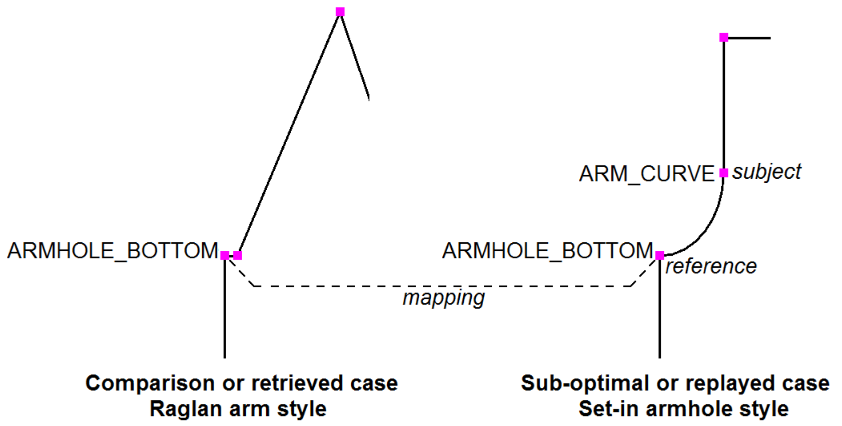


Figure 7.3: Example of a situation in which an adaptation rule will apply

Replay only occurs when a mapping exists between both {p^C, p^S} and {p^C, p^R}, otherwise (where there is no such mapping between the three shape points), the locations remain unchanged.

In reality, the *rules output case* and *replayed case* are not separate objects; the *rules output case* becomes the *replayed case*. This is accomplished by building up a list of proposed moves (LPM). Each proposed move is from p^S to p^P. The remainder of this chapter explains how the entries are added to the LPM , and then how it is checked for validity. If it is deemed valid, then the p^P locations prevail and adaptation succeeds; otherwise the p^S locations are retained and adaptation fails.

7.2.3.3 Adaptation rules

Within the context of SEACOP's implementation of CBR[12], an adaptation rule is something that prevents distortion which can arise because of a partial match of the features between the *comparison case* and *rules output case*. As previously discussed, cases in SEACOP are heterogeneous and in order to ensure that the *partial changes possible assumption* (described in section 7.1.3.1) holds, sometimes additional changes have to be added to the list of proposed moves (LPM).

Figure 7.3 shows an example in which an adaptation rule is applied. The *comparison case* is shown on the left with a Raglan armhole style. The *rules output case*, however, has a set-in armhole style. There is nothing in the criteria for mapping in adaptation (as listed in section 7.2.2.2) which prevents the armhole bottom point from being mapped. However, the arm curve point cannot be mapped since it has no counterpart in the *comparison case*. Therefore, without the adaptation rule, when the armhole bottom is moved as a result of RDR, the curve of the armhole is in danger of being distorted.

The adaptation rule used in this situation is a *coordinate difference preserver* (CDP). The effect of a coordinate difference preserver is to ensure that one of the coordinates of its *subject* Shape Point remains the same distance from the corresponding coordinate of its *reference* Shape Point. In the example in figure 7.3, a proposed move would be added to the list (if necessary) to ensure that the signed difference

[12] This definition of "adaptation rule" used here is more specific than those used in the general CBR community.

Piece type	Subject	*Reference*	*Coordinate*
Sleeve	RAGLAN_TOP	CROWN	Y
	RAGLAN_BOTTOM	CURVE_EDGE	Y
	TOP_OF_SHAPING	CURVE_EDGE	X
	BOTTOM_OF_SHAPING	LEFT_EDGE	X
Body (front, back)	ARM_CURVE	ARMHOLE_BOTTOM	Y
	YOKE_BORDER_LEFT	ARMHOLE_BOTTOM	Y
	BOTTOM_BORDER_LEFT	BOTTOM_LEFT	Y
	RAGLAN_BODY	ARMHOLE_BOTTOM	X
	FRONT_BORDER_BOTTOM	BOTTOM_OF_DIVIDE	X
	FITTED_OR_BAGGY_WAIST	BOTTOM_LEFT	X
	LEFT_EDGE	CUFF_EDGE	X

Table 7.2: Coordinate difference preservers

in the y-coordinates of the arm curve and armhole bottom Shape Points remained the same after RDR (as it was before).

A list of all the possible coordinate difference preservers is shown in table 7.2; however a CDP is only created if both its *subject* and *reference* exist. The roles which exist depend on the features in the garment; example sketches annotated with the roles are shown in appendix D.

The are two other types of adaptation rule. A *line difference preserver* (LDP) is similar to a CDP in that it also proposes a move (if necessary) to keep the distance between its subject and reference points invariant. However, the difference is that in a LDP the distance referred to is along a line, which is defined by a third point known as the *reference*. Currently, LDPs are only used to preserve the thickness of the neck band.

The final type of adaptation rule is known as a *pockets mover*. The purpose of a pockets mover is to ensure that pockets that are positioned at the edge of a cardigan front are moved, if there are proposed moves which affect the shape of the edge of the piece. For example, if the fitted waist is made narrower, then the pockets may have to be moved slightly towards the centre of the piece. The sketch instantiation algorithm has some simple rules for positioning pockets at the edge, and a pockets mover reuses these, ensuring that they take into account the changed locations of the boundary of the piece. Only one pockets mover is ever required per garment. Pockets movers affect the location of pockets, but not their size; the size is determined through reuse via RDR.

It may also be desirable to invoke move advisers during adaptation, because these can also prevent distortions that arise from heterogeneity. For example, if a proposed move would make the neck of a round neck sweater deeper, then the control points of the Bézier curve should be moved to preserve the character of that curve. If both the *comparison* and *rules output* cases have a round neck then this issue does not arise because, presumably, the control points will be moved by RDR along with the neck bottom. However, if the comparison case is a V-neck, those control points will not be present and therefore since

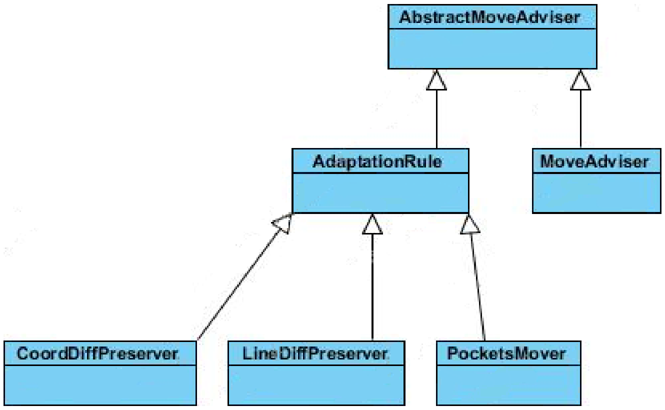


Figure 7.4: Abstract Move Adviser - hierarchy

they are not part of the mapping they are unchanged by RDR. Hence, a move adviser can be used to remedy this.

The move advisers that may apply during adaptation are those that are part of the representation of the *rules output case*; they will have been created by the sketch instantiation algorithm in the same way as if the garment was going to be created outside of CBR. Unlike the situation with adaptation rules, no special logic about features being absent in the comparison case is used in move advisers. The reason for this can be explained with reference to the example in the preceding paragraph. If both cases were round neck, then when the neck bottom was moved in the *retrieved case*, the control points would presumably be moved with it. Therefore, as the control points will be mapped to the *rules output case*, they will be part of RDR and entries will be added to the list of proposed moves for them. Move advisers will only propose new moves, they do not overwrite existing ones, and therefore in this case the move advisers will be automatically deemed not applicable.

Adaptation rules are implemented by reusing the mechanism for *exploring the consequences* (ETC), as described in section 5.2.6.2. During sketch editing, the input to ETC is the user's proposed edit; during adaptation the input is the list of mapped points with changes from RDR, as described in section 7.2.3.2.

In fact, as figure 7.4 shows, both move advisers and adaptation rules are types of abstract move adviser. Abstract move advisers have the core functionality required to participate in ETC; they can add entries to a list of proposed moves. The use of exploring the consequences allows flexibility; move advisers can be included, or CBR can be implemented with only adaptation rules.[13]

Despite having common functionality, there are differences between move advisers and adaptation rules, as listed in table 7.3. Adaptation rules have a higher priority than most move advisers as they are deemed to be particularly important during the adaptation process. The exception is *pockets mover*, which only has a standard priority; the intention is that the pockets are not positioned until changes affecting the outline of the garment have been actioned. As discussed in section 6.1, move advisers are

[13] It is always necessary to include the move advisers which preserve co-located points (which are explained in section 5.2.5.1), otherwise the integrity of the sketch may be violated.

	Adaptation Rule	*Move Adviser*
Scope	Operates during adaptation	Operates during adaptation and user editing (CAD)
Application to adaptation	Whenever the features is present in the rules output case and absent in the comparison case	When the feature is present in the rules output case.
Prioritisation	Coordinate difference preserver: high priority (1)	Co-located points: very high priority (2)
	Line difference preserver: high priority (1)	Linked points: high priority (1)
	Pockets mover: standard priority (0)	All others: standard priority (0)
	Further prioritisation is done using the notion of how much what it depends on has changed, and how many shape points it affects.	
Function in symmetry	Not applicable since adaptation only uses the editable part.	When a piece is made asymmetric by the user, the move advisers are copied, reflected and applied to the newly editable part.
Volatile (in-memory) representation	Has a temporary existence in a set that is built especially for adaptation; this set is destroyed afterwards.	An integral part of the piece.
Persistent (file) representation	Part of the program code	Stored in XML
Optionality	Not applicable	User can deactivate them

Table 7.3: Differences between adaptation rules and move advisers

an integral part of the structure of a garment. However, adaptation rules only exist during adaptation; they are relatively few in number and so are hard-coded into the Java program, for simplicity.

The list of proposed moves will consist of entries added by RDR, as explained in section 7.2.3.2. In addition, further entries will be added by adaptation rules and move advisers, in order to preserve integrity of certain key features. The addition of these further entries is automated by *exploring the consequences*; thus, a key algorithm from the CAD aspect of SEACOP is reused to assist with adaptation.

7.2.3.4 Checking and failure of adaptation

The output of the processes discussed previously is the *rules output case*, and a *list of proposed moves* (LPM). The next stage is to apply the moves in the LPM to the *rules output case*, to produce the *replayed case*.

Before this is done, however, a check is carried out to ensure that the LPM would not introduce any violations of the fundamental constraints. There are two types of violation: crossovers and non-intersect; these are explained in section 5.2.6.5. The algorithm which checks this is the same one that is used for movement (see section 5.2.6.4). However, there is a key difference: violations of fundamental constraints which may occur during adaptation are not fixed in the same way as they are during movement. This difference is deliberate: fixing violations uses heuristics and during movement, the user receives visual feedback of the effect of these. Adaptation is different as it is a (semi) automated process and the user would not be able to differentiate between changes that have occurred during RDR, changes from abstract move advisers, or those from fixes.

If the fundamental constraints check is not passed, then adaptation has failed - this is referred to as a *technical failure*. Otherwise, the result is presented to the user and a subjective judgement is made as to whether adaptation is a *success*, or a *user-decided failure*.

If the result is a *technical failure* or *user-decided failure*, then CBR can move on to retrieving another case from the case base. Alternatively, the algorithm can output the *rules output case*; the user is then free to edit this manually to produce the *edited case*. Thus, the hybrid nature of the system has provided the implementer with flexibility in how to deal with adaptation failure: there is the option of using the rules as a fall-back.

Chapter 8

Experiments

"All of science is nothing more than the refinement of everyday thinking." - Albert Einstein

This chapter describes some experiments that were conducted using the algorithms described in chapters 6 and 7. The experiments are designed to establish how the algorithms fit into the theoretical framework of CBR (as presented in the literature), to gather evidence to describe how the algorithms may be used in practice, and to show how they may be improved through further work.

Section 8.1 outlines some questions which are relevant to the way in which CBR can be used in SEACOP. Section 8.2 explains how an experiment was devised to help answer these questions, and the results of this experiment are shown in section 8.3. All experiments have limitations, and a simulation that overcomes some of these limitations is presented in section 8.4.

8.1 Hypotheses

This section details the hypotheses that the experiments are designed to prove (or disprove); each hypothesis is then discussed below.

1 CBR will produce designs that are more often accepted by the domain expert (than rules alone).

2 The use of all move advisers will be the best mode of operation.[1]

3 The minimum average distance (MAD) can be used as an indication of quality.

4 As the distance between retrieved and query cases increases, the effectiveness of CBR will decrease.

5 Questionnaire similarity can be used as a guide to whether to invoke CBR or not, i.e. as an estimate for the success or otherwise of CBR.

6 As questionnaire similarity increases, the amount of change in adaptation will decrease.

7 The performance of SEACOP will improve as the case base grows, good performance will be achieved with a few hundred cases.

8 SEACOP will normally be capable of producing output within an acceptable time (less than a minute).

Obviously, where any research involves the creation of new techniques or algorithms, an important part of that research will be to test their effectiveness. SEACOP is no exception, and therefore the central hypothesis being tested is that the output of CBR should outperform rules (hypothesis 1). Rules are inflexible, and in some domains such as fashion design they are often interpreted liberally in practice. The hypothesis implies that the CBR has the flexibility to respond to deviations from those rules.

The adaptation algorithm described in chapter 7 can be used in two main modes: by invoking all move advisers in the replayed case during the adaptation phase, or using only adaptation rules.[1] These two modes of operation must be contrasted to determine their relative effectiveness. Hypothesis 2 arises from the belief that the use of move advisers will improve the output by enforcing constraints.

In order to answer questions about performance, it is necessary first to decide how quality can be measured. Some measures for sketch similarity (e.g. MAD) were discussed in section 6.3. Hypothesis 3 asserts that can can be used as an indication of quality, since it can be used to objectively compare the sketches of the output and goal cases. Increased values of MAD are expected to correspond to lower quality outputs.

The perceived relationship between similarity and the effectiveness of CBR arises from the assumption that similarity correlates with adaptability [4]. Therefore, hypothesis 4 states that as the distance between retrieved and query cases increases, the effectiveness of CBR will decrease.

These experiments are conducted using the assumption that the questionnaire similarity can be used as the measure of similarity between the retrieved and query cases. Since the sketches are derived from the questionnaire, there is clearly a link between the two, so it is reasonable to assume that similarities in the questionnaire will correlate to similarities in the sketch, and these in turn will correlate with adaptability.

If the assumption about questionnaire similarity holds, and hypothesis 4 is true, then it is also expected that the questionnaire similarity can be used as a guide to whether to invoke CBR or not (hypothesis 5). If the similarity falls below some preset value (which is derived through experimentation), then it may be that the output of the rules can be expected to be better than that of CBR.

Hypothesis 6 arises from the intuitive response that the more similar two objects are, the less change should be required to turn one into another. However, it is uncertain how this hypothesis fits into the RDR paradigm.

[1] And level 2 move advisers for co-located points.

The relationship between case base size and effectiveness is a popular theme in CBR research [173]. Assuming that there is sufficient coverage of the case base, hypothesis 7 is concerned with SEACOP's performance (as defined by 'excellence' in section 8.4.2) when the case base has grown to a few hundred cases.

Efficiency in the knitwear domain is not as important as in others, since the existing manual processes (see section 5.1.1) are slow and the emphasis is on quality rather than productivity. However, it is still important that the system provides an output within a reasonable amount of time, in order to obtain the acceptance of users. Hence, hypothesis 8 states that SEACOP must be capable of producing output within an acceptable time.

8.2 Methodology

Section 8.2.1 explains how the case base was built up. 'Seeding' the case base is a common issue in CBR, but fortunately in SEACOP the output of the rules can be used to assist with this. Section 8.2.2 explains how the automated experiments were designed, and section 8.2.3 describes how the results were evaluated.

8.2.1 Case base

In order to evaluate SEACOP, it is necessary to first populate the system with a test case base. The composition of this case base is given in appendix E. It was possible to obtain 26 cases from the sponsors of this research, Sirdar Spinning Ltd. Twelve designs were obtained from two knitting books [6, 174]. A further twelve designs were created by myself.

In total, the case base is populated with 50 garments. Maximising the number of garments is important for CBR research, as it is well known that performance increases with case base size, at least for small case bases. In [173], a study was conducted on the effect of case base size on the accuracy of CBR using four different case bases; the results were presented as line graphs. Using 50 cases appeared (on manual inspection of the graphs) to achieve between approximately 70 to 85% accuracy, dependent on the case base and experimental method chosen. The shape of the graph indicated that performance increases very rapidly with case base size for small case bases (i.e. less than 20 cases), but the improvements gradually decrease until the graph reaches an asymptote. In all four case bases, 50 cases appeared to be close to the optimum value (which was less than 100% accuracy). Of course, results will vary from domain to domain.

Several factors make it difficult to gather cases, e.g. it was only possible to obtain a limited number from the sponsor. Knitwear literature can be another sources of cases. However, the majority of knitting books concentrate on topics such as basic techniques in knitting. Sketches are the focus of this research, and very few knitting books or patterns feature sketches, as (understandably) they are aimed at knitters, and sketches are deemed to be unnecessary, because they do not explain how to knit the garment. Also, sketches are thought of as too complex or confusing for ordinary knitters, because a 2-dimensional sketch does not always correspond exactly to the appearance of a 3-dimensional garment.

So, in order to obtain sketches from knitting books or patterns, a process of reverse engineering is usually necessary. This involves translating the written instructions into measurements. Sometimes the measurements are stated explicitly in the pattern, but often only a number of stitches is quoted and the measurement is obtained using the tension. From the measurements, a rough drawing is made using pencil and paper. Then, inconsistencies in the measurements are resolved; these inconsistencies arise from rounding errors in the instructions. Then, the questionnaire attributes are entered into SEACOP and the resulting sketch is edited to ensure it is consistent with the drawing. The consistency referred to is in the measurements and not the shape: it is unlikely that the rough drawing will be perfectly accurate. It typically takes three hours to create a case using this process.

8.2.2 Design of automated experiments

The design of these experiments is different to the design of a system which would be used in practice, since in the former it is desirable to collect data from which the research questions and hypotheses discussed in section 8.1 can be addressed.

The main[2] experiment was conducted as follows. The garments that comprise the case base were loaded into volatile memory. The questionnaire similarities of each pair of garments were computed and saved to a file.

Then, each garment from the case base was obtained and used as the query case. For each query case, adaptation was attempted with all the other garments in the case base. This is a form of leave-one-out testing, which has been utilised in CBR research by others, e.g. [173]. The key advantage of the leave-one-out methodology is that it allows all the cases to be used, which is important when the case base has a limited size (as per section 8.2.1). Each adaptation attempt was given one of three category labels:

- *Failure:* if adaptation was a technical failure (see section 7.2.3.4).
- *Success*: if adaptation was not a technical failure, but another successful attempt had a higher similarity than this.
- *Retrieved*: otherwise, i.e. if adaptation was not a technical failure, and the questionnaire similarity was greater than any other for this query case. For each retrieved case, the sketch similarity

[2] The efficiency experiments and simulation had different methodologies which were explained in sections 8.3.6 and 8.4.2 respectively.

measured using MAD (see section 6.3) to the goal was recorded.

The experiments were performed twice, once using all move advisers during adaptation, and once using only the level 2 move advisers (which are necessary to maintain the integrity of the sketch). Thus, the use of move advisers in adaptation, which was explained in section 7.2.3.3, was explored.

The results from both invocations were stored as separate XML files. Additionally, results were also produced as image files. Each image file contains the sketch of the output superimposed on the goal. Up to three images per query case were produced: one for the output of the rules, and the other two for the modes of CBR (as discussed above). The superimposition is non-trivial since the two garments are not necessarily composed of the same types of pieces (e.g. one may be sleeveless and the other not), and the pieces each have their own coordinate system. The superimposition works by translating each piece until their coordinates coincide. The bounding box of each pair of pieces is calculated, and a heuristic is used to arrange these within the image. This heuristic algorithm is the same one used to arrange pieces in the sketch editor, as depicted in figure 5.1. Thus, an algorithm from the computer-aided design part of the project is being reused to implement the experiments.

8.2.3 Evaluation of results

The evaluation of the quality of the output of CBR is important, in order to be able to judge the effectiveness of the system as a whole, and to make decisions about how it can be used in practice. The MAD between the goal and output may provide an objective measure of the quality of the output. However, MAD has several shortcomings. Firstly, it does not take into account differences in the composition of the pieces; if a Shape Point is present in one garment, and there is no corresponding object in the other, then it is ignored.

Secondly, MAD is an unsophisticated measure of similarity. As an illustrative example, consider the situation in which the origin of a sweater front is moved down by 10cm. As a consequence, move advisers would subject the origin of the back and the bottom left of both pieces to the same translation. Thus, the back and front would be lengthened by 10cm, which is likely to result in a large MAD between the two garments. However, the edit is easy to revert, so clearly MAD does not always correlate with the difficulty of editing (as discussed below).

In addition to MAD, the results were scored subjectively (by a human) using a Likert [64] scale, as shown below. Various sources were considered for the category labels, e.g. [175]. The words (below) were chosen because they were descriptive, and each label consists of only one word (unlike, e.g. "very good"). Also, despite its appearance in the literature, "average" was not used here as a label as it carries an implication of statistical significance, which could be misleading.

1 *Bad*: although a technical success (in the sense discussed in section 7.2.3.4), the result is probably

not feasible as a garment. It may have shapes that are difficult to knit, and that would not form part of a practical, wearable garment.

2 *Poor:* has some good features but also significant differences; the result is unlikely to be considered a close match to the goal.

3 *Fair*: could be converted into the goal, but would require several non-trivial edits.

4 *Good*: is recognisably similar to the goal.

5 *Excellent*: is very similar or identical to the goal; only one edit, or a small number of edits are likely to be required.

Only integer scores were permitted, and when determining the score several factors were taken into account, as listed below. The factors are not mutually exclusive and were not used in isolation: good performance in one area may make up for poor performance in others. In approximate descending order of importance, the factors are:

- *Difficulty in editing:* the effort required to transform the result into the goal is relevant; if the output is something which is difficult to edit into the goal, it will receive a low score. For example, it is easier to change straight lines than curves. This is the most important factor, since it is likely that the user will be involved in correcting the output of CBR when it is not judged that improvements are required.

- *Number of errors:* the number of deviations in the positions of the points is important. It is easier to correct one large error than 3 small errors, for example.

- *Correctness of shapes:* the result should have similar angles and curves to the goal.

- *Distances:* the difference between the locations of equivalent Shape Points in the output and goal are relevant. This is what is being measured in MAD, but it is an aspect of the subjective score also. If the distances are high, the two shapes will appear different and so the output will tend to receive a lower score.

- *Size of the pieces:* differences between the areas of the bounding boxes are relevant.

In order to apply these criteria the human judge would not need to be an expert, but would require some specialist knowledge. Apart from the obvious visual ability to distinguish shapes, they need a some knowledge of knitwear design, for example, so that they recognise that a sketch is corrupted (e.g. as shown in figure 8.8). Also, they need some experience in the sketch editing software, which is described in chapter 5. This is necessary in order to fully appreciate the relative difficulty of making edits. For

Score	*CBR*		*Rules*	*All*
	All move advisers	*Only level 2 move advisers*		
failure	2	6	0	8
1	4	9	1	14
2	18	19	15	52
3	13	13	20	46
4	10	2	10	22
5	3	1	4	8
Total	50	50	50	150
Mean (2 d.p.)	2.79	2.25	3.02	2.70

Table 8.1: Distribution of Scores: CBR versus rules

example, an evaluator may be experienced in other software which allows the users to manipulate curves directly, which is not the case in SEACOP (which uses control points). Due to the difficulty in obtaining people with such experience and knowledge, it was decided that I should evaluate the results myself. The obvious disadvantage of this is the lack of impartiality. However, as the examples in section 8.3.2 show, I believe the scores given can be justified by the characteristics of the result, according to the criteria listed above.

8.3 Results

The results of the experiments are presented in this section. Unless stipulated otherwise, non-integer results are rounded to 4 decimal places.

8.3.1 Hypotheses 1 and 2: CBR compared with the rules

The results of CBR are compared with the rules to test hypothesis 1, that the use of CBR should outperform rules. Also, it is possible to contrast the two modes of operation of CBR (hypothesis 2).

Table 8.1 and figure 8.1 show the distribution of scores where CBR is run in two different modes, compared to the rules alone. The topmost row in table 8.1 indicates the cases that failed and thus were not scored. The remaining rows show the number of cases which were given the score shown in the column on the left hand side. The bottom two rows show summary information; the mean score is rounded to two decimal places. The total number of results is 150, as each of the 50 cases was run in the three different modes (as indicated by the column headings).

The mean score for CBR is lower than that of the rules. However, the mean score with all move advisers is higher than those with just level 2 move advisers, being closer to that of the rules. Therefore, all results presented in subsequent sections as being CBR will relate to the all move advisers mode.

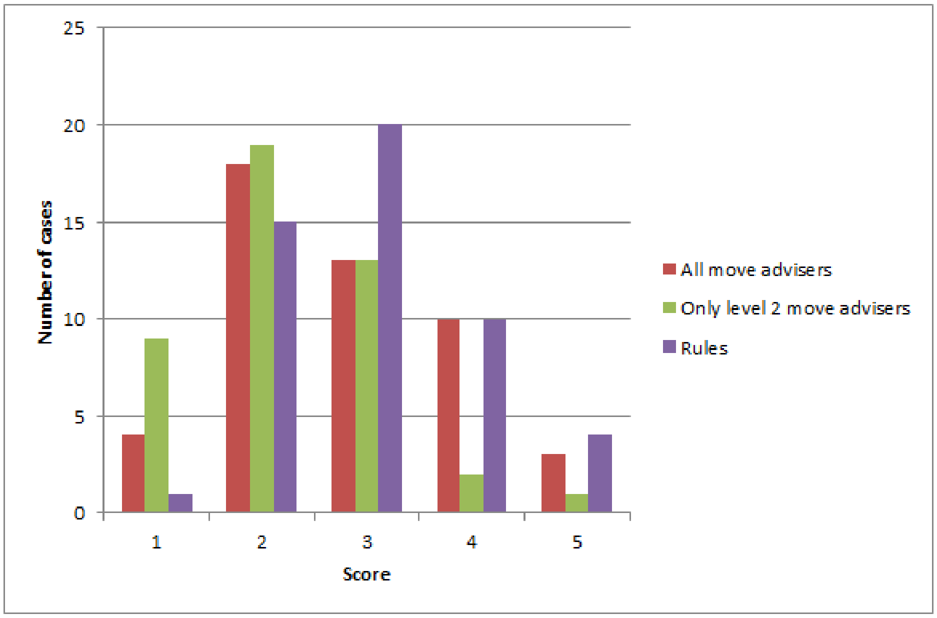


Figure 8.1: Distribution of Scores: CBR compared to Rules

Table 8.2 shows the scores grouped by the source of the query (or goal) case, for both CBR and rules. Each column represents one source. The values in the cells refer to the number of cases which were given the score shown in the leftmost column. The results for the rules are all fairly similar across the sources, with the exception of the 'PR' cases, where the mean[3] scores of the rules are significantly higher than the others. This suggests that in those cases, the goal is probably fairly similar to the rules output. In the CBR results, the Budd cases are higher than the overall mean and the Badger ones lower, but as the sample sizes are quite low there may not be significant.

8.3.2 Examples

In this section, selected examples of adaptation are presented. Three examples were chosen: each one is representative of an excellent, fair or poor result. Specific examples help to illustrate the decision making during the scoring process, provide some intuitive evidence of the accuracy of the questionnaire similarity algorithm, and examine how RDR works in more detail than can be provided by statistics alone. The important attributes of the cases in question are listed in appendix E.

8.3.2.1 budd58b: an excellent result

With budd58b as a query case, the most similar other case was budd58c, with a questionnaire similarity of 0.9596. Since the two questionnaire are the same except for the neck style and depth, the very high similarity value is as expected.

Adaptation was technically successful. The result shown in figure 8.2 indicates a perfect match on

[3] Mean scores were rounded to 2 decimal places.

Score	Sirdar	*PR*	*Budd*	*Badger*	*All*
1	1	2	0	1	4
2	13	3	0	2	18
3	4	5	3	1	13
4	5	1	4	0	10
5	1	1	1	0	3
Total	24	12	8	4	48
Mean (2 d.p.)	2.67	2.67	3.75	2.00	2.79

(a) CBR

Score	*Sirdar*	*PR*	*Budd*	*Badger*	*All*
1	0	0	0	1	1
2	10	0	4	1	15
3	14	0	4	2	20
4	1	9	0	0	10
5	1	3	0	0	4
Total	26	12	8	4	50
Mean (2 d.p.)	2.73	4.25	2.50	2.25	3.02

(b) Rules

Table 8.2: Scores grouped by source of garment

the sleeve and back, and only two defects on the front: the neck band is too narrow and the neck is too deep. However, these defects are easy for the user to remedy, and because of the perfect sleeve and back the output of CBR scores a 5.

The adaptation replays 26 changes to the garment. The only Shape Points in the garment that are unmapped are the two neck Bézier control points. However, these have been set to appropriate positions by move advisers in the *replayed case*. Thus, distortion that otherwise would have occurred (since the retrieved case has no Bézier curves in the front) has been prevented by the use of move advisers. This constitutes evidence in favour of hypothesis 2.

The defects arise because the V neck in the retrieved case has a narrower band and deeper neck. It would be possible to create new conditions on mapping (see section 7.2.2) which forbid reuse of changes to the neck bottom point. However, the more conditions that are added, the less adaptation takes place so the composition of the conditions is a balance between control and reuse. With such an additional condition in place, the neck bottom point on the front would be as per the rules, and there is no guarantee that this would be better, as figure 8.3 shows.

8.3.2.2 s3014: a fair result

With s3014 as a query case the most similar other case was s3003, with a questionnaire similarity of 0.8406. The differences in the questionnaire attributes relate to the bottom border, neck style and tensions. The similarity value, which is high but not exceptionally high, seems sensible given these differences.

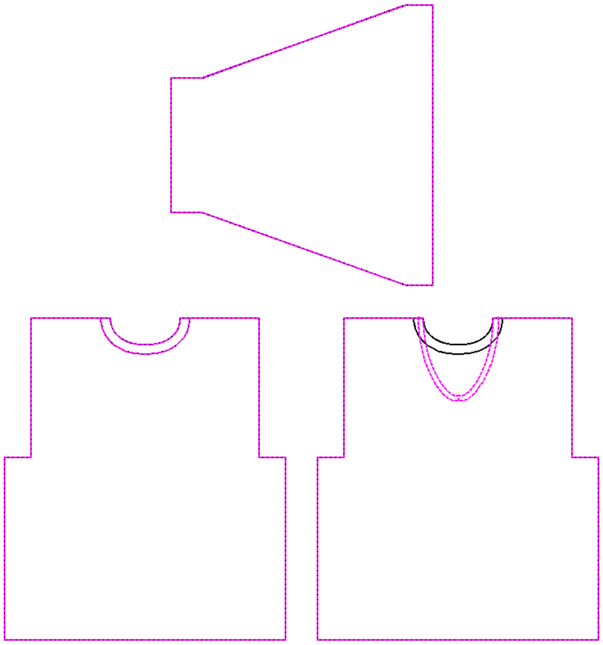


Figure 8.2: budd58b: CBR result in magenta, goal shown in black

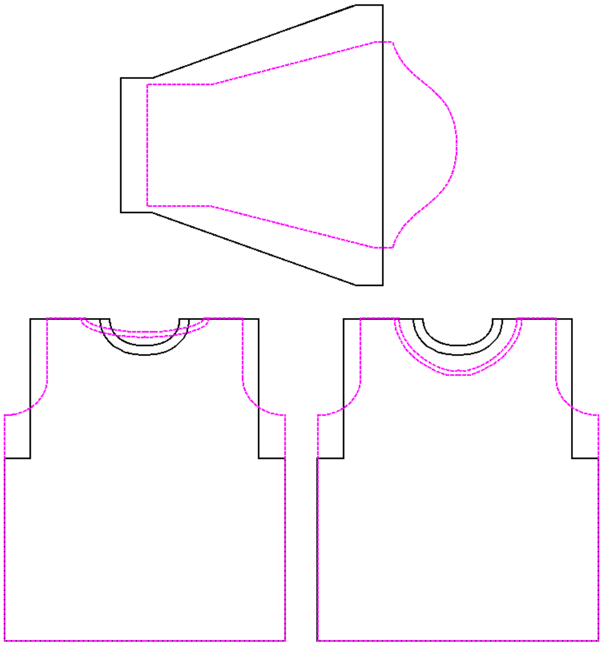


Figure 8.3: budd58b: rules output in magenta, goal shown in black

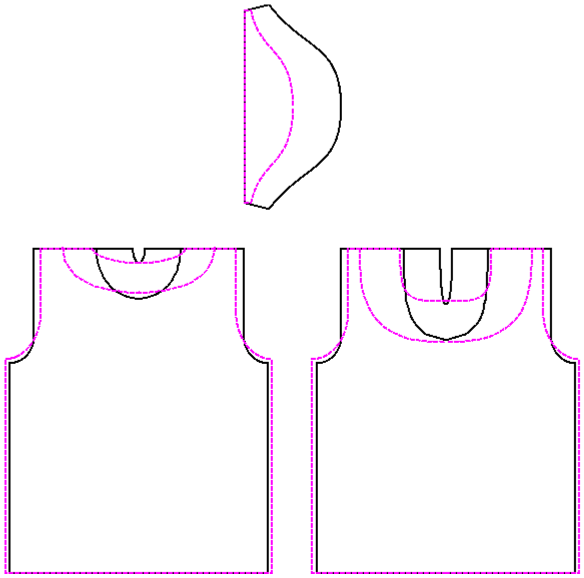


Figure 8.4: s3014: CBR output in magenta, goal shown in black

Adaptation was technically successful and the results are shown in figure 8.4. The results bear some resemblance to the goal, and although there are many differences, the sizes of the pieces are similar, so it was scored as a 3.

The result of the rules is shown in figure 8.5. The sizes of the pieces are a poor match (except for the height of the body). The neck band width is also poorly matched. Therefore, it scored a 2.

The most interesting part of the adaptation of s3014 is the sleeve length. SEACOP has four options for sleeve length; in order of increasing length they are: very short, above elbow , three quarter length, and wrist. However, despite its name, the "very short" sleeve length is not the shortest possible (realistic) sleeve. Ideally there should have been another option, possibly called "cap sleeve" which was shorter still. However, this requirement was missed at the modelling stage. In many systems with a more rigid architecture, this would cause ongoing problems for users. However, SEACOP has replayed the shortening of the sleeve from s3003, as shown in figure 8.6.

8.3.2.3 s3041: a poor result

When s3041 is the query case, adaptation fails with the 41 most similar cases. The most similar successful case is s3011, which has a similarity of 0.3975. There are many differences between the two cases, so the low similarity is as expected.

The result was technically successful and is shown in figure 8.7. The bands and borders in the output are too narrow, the sleeve length is too short, the front is too narrow and it has not captured the shape of the front well, hence it was scored as a 2.

The goal case in s3041 is a garment which is on the limit of SEACOP's capabilities. It is shown in figure 8.7 as having a separate front border, bottom border and neck band; in reality these were not

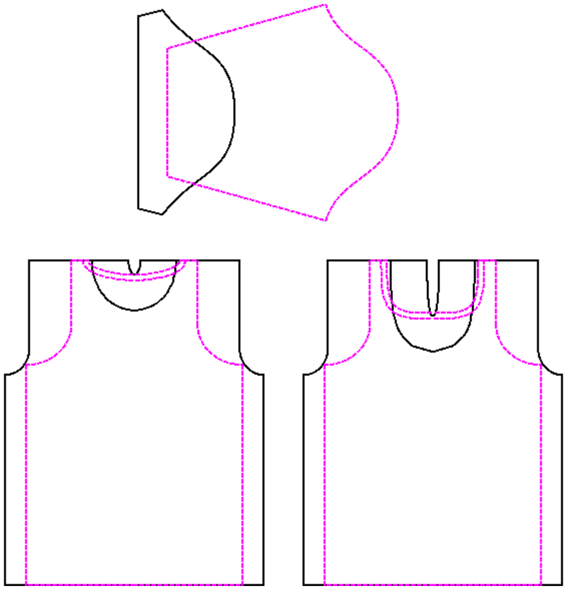


Figure 8.5: s3014: rules output in magenta, goal shown in black

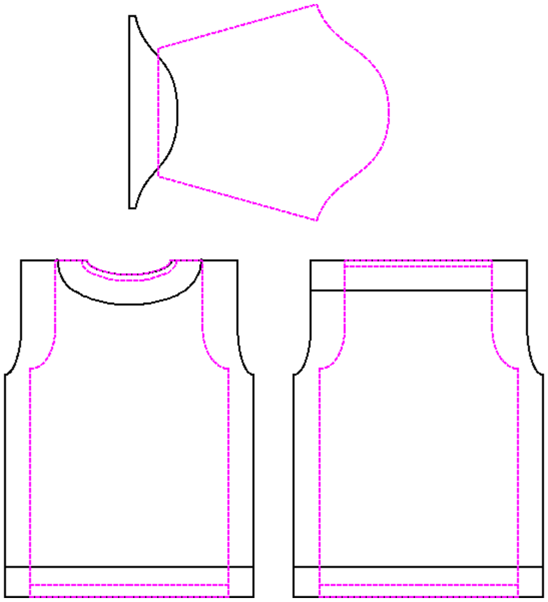


Figure 8.6: s3003: rules output in magenta, goal shown in black

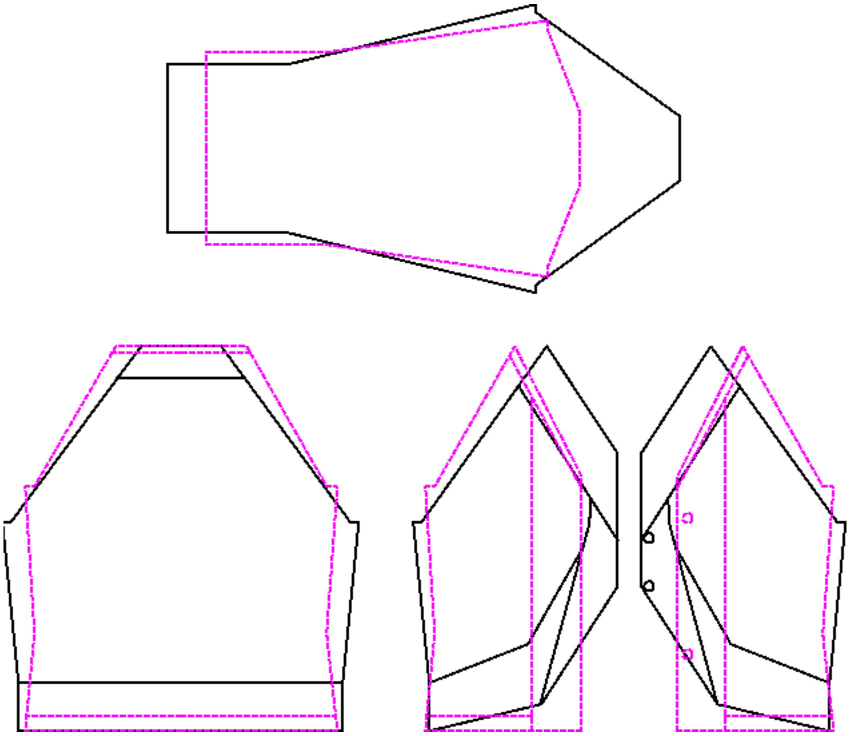


Figure 8.7: s3041: CBR output in magenta, goal shown in black

separate and formed one long continuous curved border (an option which is not available in SEACOP). The design utilises user-added points, and these are never created during adaptation. The complexity of the query case is presumably the reason why it is so difficult to adapt.

Figure 8.8 shows the result of adaptation where the retrieved case is s3311, which is the most similar (with a questionnaire similarity of 0.8400). Adaptation was deemed to have failed because the neck of the back is corrupted.

s3041 is a noteworthy example because of the low similarity rank of the retrieved case. The garment in the case base which has the greatest questionnaire similarity to the query case (other than the query case itself) is assigned a rank of 1. The ranks then increase in descending order of questionnaire similarity[4], until the garment with the lowest similarity which has a rank of 49. As table 8.3 shows, the majority of adaptations (37 out of 48) succeed with one of the two most similar cases. s3041 is the most extreme example in the case base, with a retrieved rank of 42 (this is the row in the table that is third from bottom).

Figure 8.9 shows a scatter-plot of adaptation attempts with s3041 as the query case. Each point represents an adaptation attempt (which is not necessarily successful). The horizontal axis shows the questionnaire similarity between s3041 and the retrieved case. The vertical axis shows the number of changes that are made to the replayed case, after exploring the consequences (ETC) has been invoked. The red diamond indicates the datum corresponding to s3011 (the retrieved case with the highest similarity

[4] Ties (where two different garments have the same similarity value with respect to a third garment) are very rare.

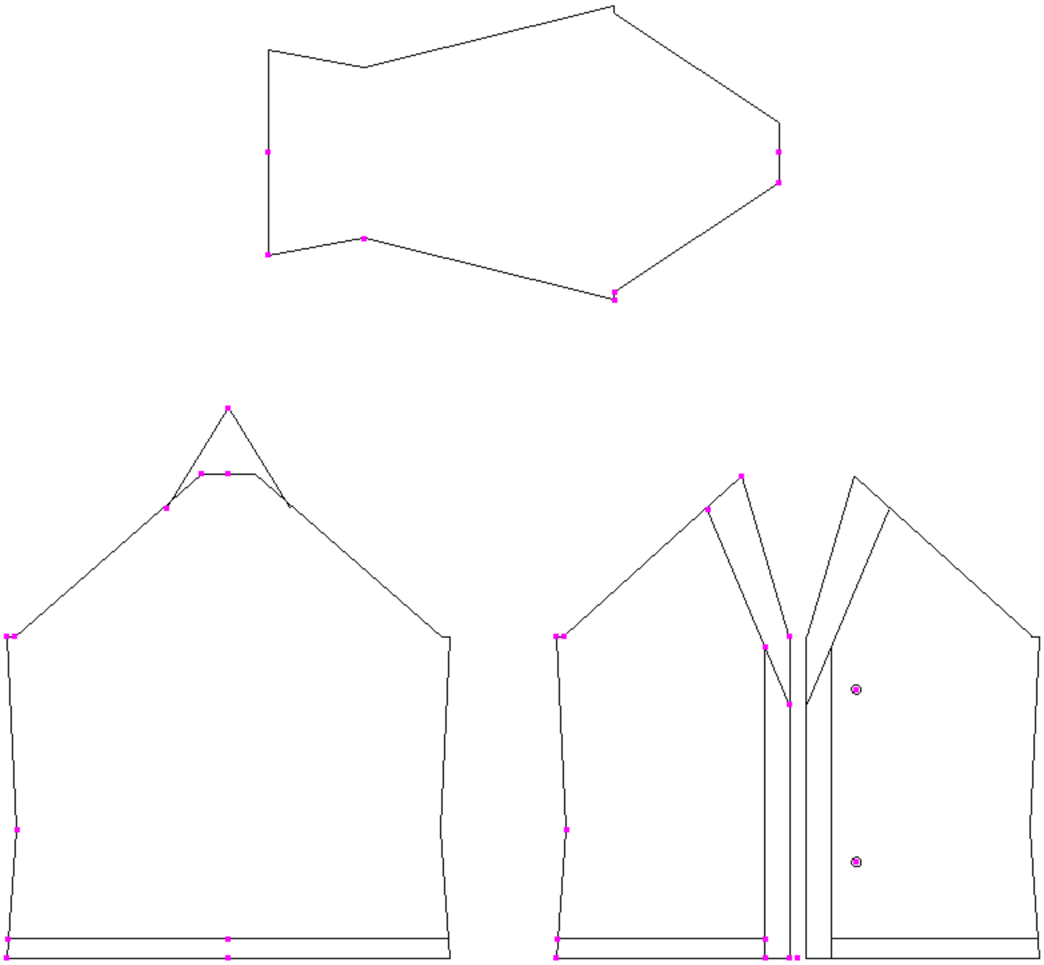


Figure 8.8: Replayed case where s3041 is the query, and s3311 is retrieved

Rank	*Number of query cases*
1	28
2	9
3	1
4	2
5	3
7	3
8 to 14	0
15	1
16 to 41	0
42	1
43 to 49	0
Total	48

Table 8.3: Distribution of ranks of questionnaire similarity of retrieved cases

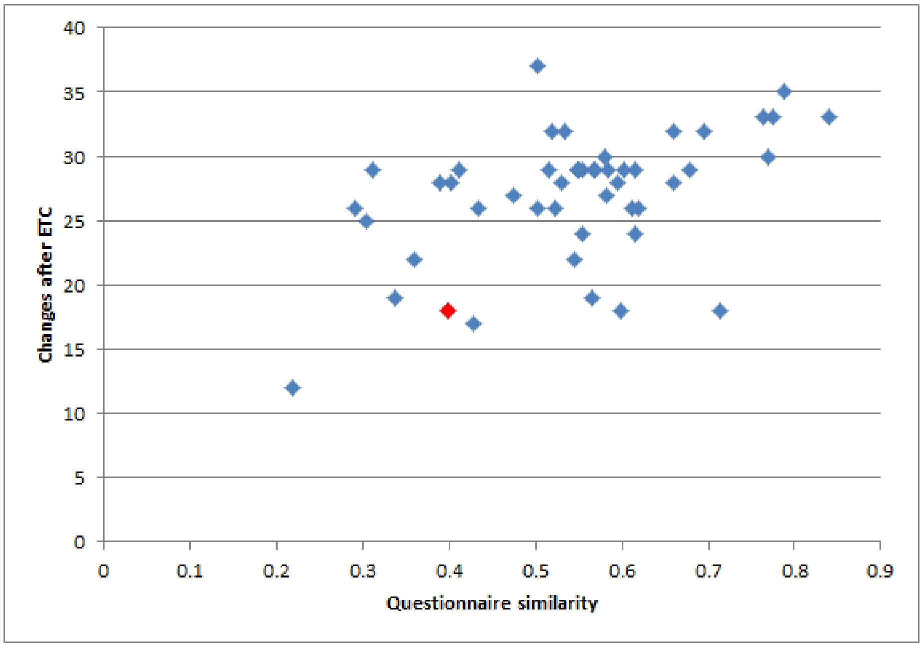


Figure 8.9: Changes in s3041 after ETC versus questionnaire similarity

which was associated with successful adaptation).

The data depicted in figure 8.10 has a Pearson's r of 0.4816. The level of significance for this (two-tailed test for 50 cases) is 0.0004. This indicates that there is a positive correlation between questionnaire similarity and the number of changes, in the case of s3041. Section 8.3.5.1 shows similar data for the case-base as a whole. The significance of this is discussed in more detail in section 8.5.1.

8.3.3 Hypothesis 3: Comparing score and MAD

The objective of comparing scores and MAD is to determine whether MAD can be used as an indication of quality, as per hypothesis 3. Figure 8.10 shows a scatter plot of the score versus the MAD between the goal and result. The graph shows a negative correlation: higher scores are associated with lower values of MAD. Also, with the exception of the 4 cases which scored 1, the range of MAD values decreases as the score increases. The Pearson product-moment correlation coefficient (Pearson's r) is -0.4206, with significance level of 0.0024 (two-tailed test for 50 cases). Therefore, the negative correlation between score and MAD is very unlikely to have occurred by chance.

The question can be formulated: what score is typically associated with a given questionnaire similarity? Figure 8.11 shows a scatter plot of Score versus MAD, with a logarithmic trend-line added. Other functions (exponential, linear, power and polynomial) were tried for the trend line but all had a higher error rate, with the exception of a quintic function.

If the logarithmic function shown in figure 8.11 is rounded to the nearest integer, then an estimated value for the score can be obtained from the MAD. Table 8.4 shows the distribution of absolute differences between the projected and actual scores. In 88% of cases, the estimated score is within ±1 of the actual

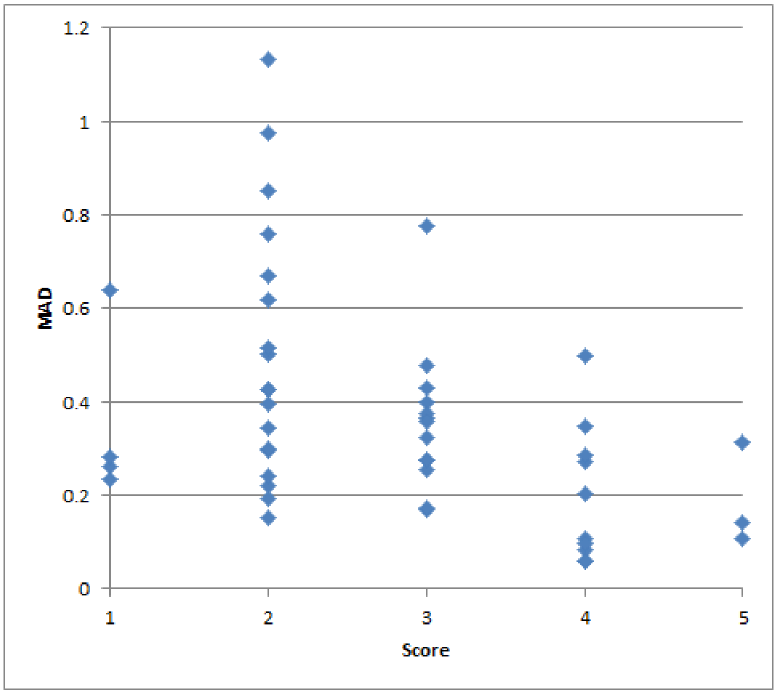


Figure 8.10: MAD versus score

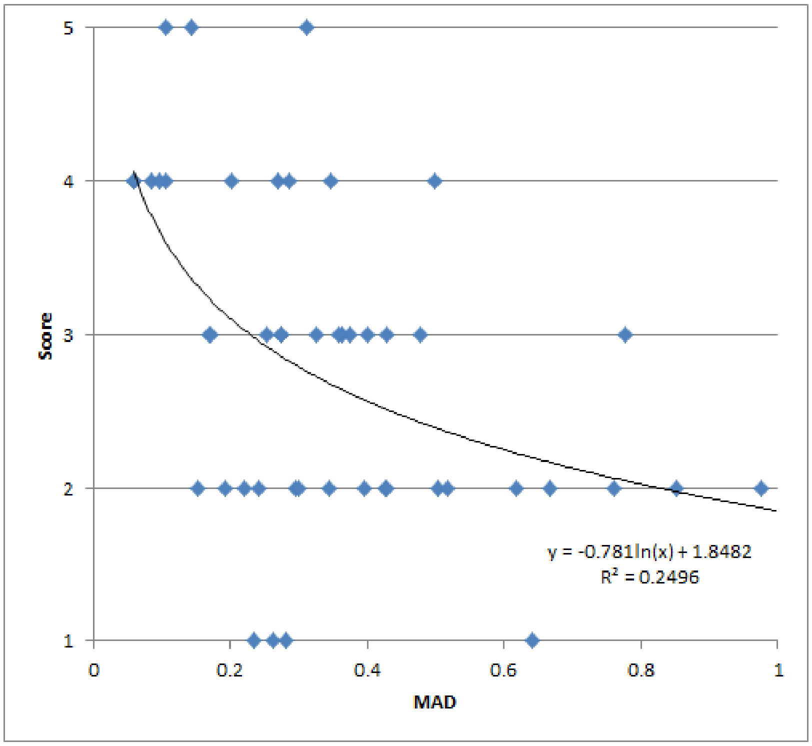


Figure 8.11: score versus MAD, with trend-line

Difference	*Cases*
0	24
1	18
2	6
3	0
4	0

Table 8.4: Distribution of differences between estimated score based on MAD, and actual score

score. The significance of this is discussed further in section 8.5.1.

8.3.4 Hypothesis 4: Score and questionnaire similarity

The purpose of comparing questionnaire similarity with score is to investigate the relationship between similarity and quality, as per hypothesis 4. Furthermore, the use of the questionnaire similarity as a guide in retrieval (hypothesis 5) can be investigated.

8.3.4.1 Comparison

The scatter plot in figure 8.12 compares questionnaire similarity (between query case and retrieved case) and the score. The graph looks similar to a vertical reflection of figure 8.10. There is a clear positive correlation between questionnaire similarity and score, although there are a few outliers such as the one with a score of 5 and a questionnaire similarity circa 0.65. Pearson's r = 0.5793, at the significance level 0.000038 (to 2 significant figures; two-tailed test for 44 cases, as the 6 failures are not shown). Therefore, the correlation between score and questionnaire similarity is extremely unlikely to have occurred by chance.

8.3.4.2 Using the questionnaire as a guide

Table 8.5 and figure 8.13 show the distribution of cases divided into three categories: where the CBR (with all move advisers) has the maximum score, where the rules have the maximum score, and where there is a tie.[5] The results show that the majority of cases in which CBR gives the best output are where the questionnaire similarity exceeds 0.93.

8.3.5 Hypothesis 6: Questionnaire similarity and the amount of change

In order to investigate the amount of change (hypothesis 6), some definitions are necessary:

- *Map size* – is the number of entries in the mapping between the query and retrieved case.

[5] If both modes of CBR are considered, then the 17 tied cases result from 5 different types of scenario. The most common of these (with 6 cases) is where the 'all move advisers' mode and the rules score are identical, but the 'level 2 move advisers only' score is lower. There are three cases in which the 'level 2 move advisers only' score is equal to that of the rules, but the 'all move advisers' score is the lowest. There are cases in which the 'level 2 move advisers only' score is better than the 'all move advisers' one; however in these scenarios, the score for the rules is always equal or better than that of the 'level 2 move advisers only' score.

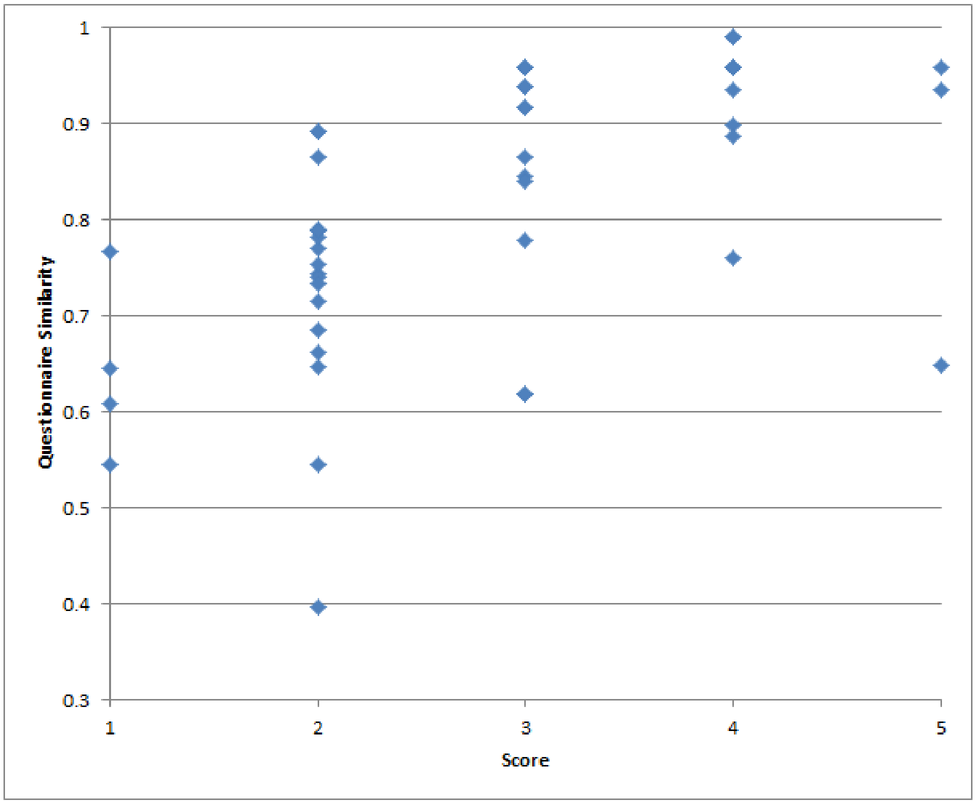


Figure 8.12: Questionnaire similarity versus score

Questionnaire Similarity Range	*CBR*	*Rules*	*Tie*
$0.3 < s \leq 0.37$	0	2	0
$0.37 < s \leq 0.44$	0	0	1
$0.44 < s \leq 0.51$	0	0	0
$0.51 < s \leq 0.58$	0	2	0
$0.58 < s \leq 0.65$	0	4	2
$0.65 < s \leq 0.72$	1	1	1
$0.72 < s \leq 0.79$	0	7	4
$0.79 < s \leq 0.86$	0	1	2
$0.86 < s \leq 0.93$	1	3	4
$0.93 < s \leq 1$	11	0	3
$0 \leq s \leq 1$	13	20	17

Table 8.5: Breakdown of results for CBR, rules and tied situations

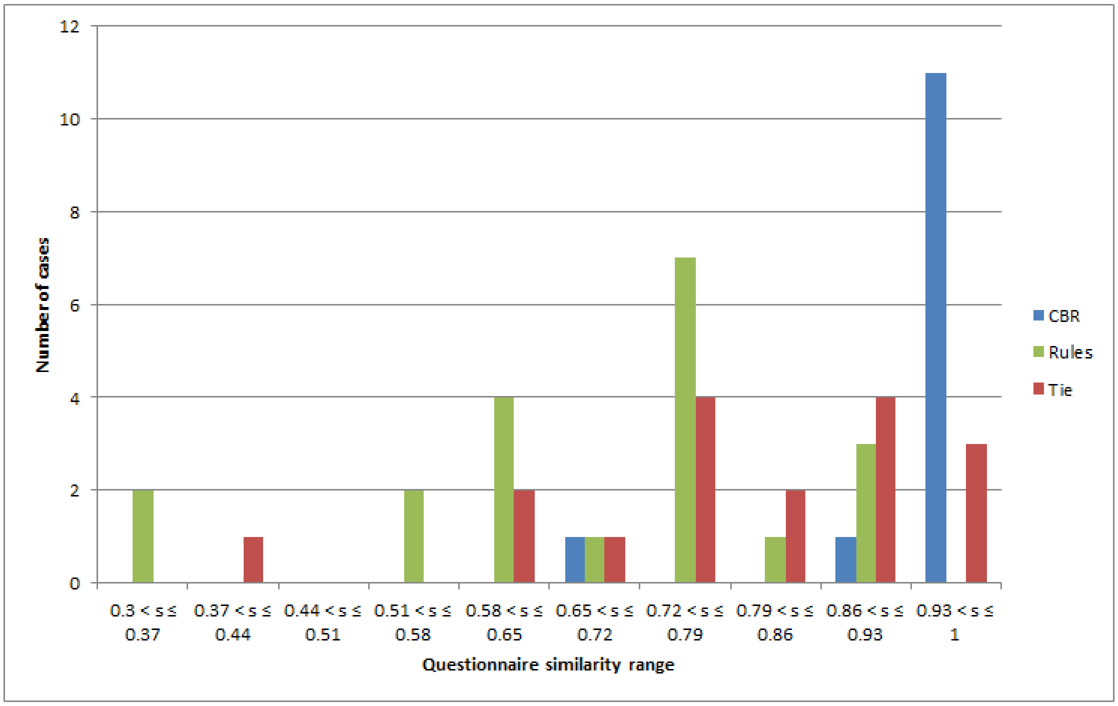


Figure 8.13: Breakdown of results for CBR, rules and tied situations

- *Changes* – is the number of changes in the replayed case brought about directly because of the map.
- *Changes with ETC* – the changes after exploring the consequences has been invoked.

Hence:

$$map\,size \leq changes \leq changes\,with\,ETC$$

Null change refers to the situation where there was a mapping which did not result in a change, so:

$$null\,changes = changes - map\,size$$

The proportion of null changes refers to the proportion of map entries which result in no change, i.e.:

$$proportion\,of\,null\,changes = \frac{null\,changes}{map\,size}$$

8.3.5.1 Map size, changes and changes with ETC

Figure 8.14 shows scatter plots of map size, changes and changes with ETC versus questionnaire similarity. Each point in the scatter plot relates to one of the 2450 unique pairs of garments in the case base. Figure 8.14a shows a clear positive correlation between questionnaire similarity and map size. The shape of figure 8.14b is similar but with more of a spread, indicating a positive (but reduced) correlation. The

Datum	*Pearson's R*	*Significance (two-tail)*
Map size	0.6032	<0.0000001
Changes	0.5189	<0.0000001
Changes with ETC	0.0238	0.2390

Table 8.6: Correlations with Questionnaire Similarity

spread is larger still in figure 8.14c, which shows no obvious correlation from a visual inspection. These observations are consistent with the correlation data in table 8.6.

Figure 8.15 visualises the same data in a histogram; this confirms that map size and changes clearly increase as similarity increases. However, the changes with ETC values show no such correlation and are relatively constant at around 25-30.

8.3.5.2 Proportion of null changes

Figure 8.16 shows a scatter plot of the proportion of null changes versus questionnaire similarity; no obvious relationship is apparent. However, when the data is summarised in figure 8.17, it looks as if there is a possible weak negative correlation.

8.3.6 Hypothesis 8: Efficiency

In this section an investigation into how long SEACOP will take to typically produce an output (hypothesis 8) is presented. Some indicative times taken to perform specific tasks are given below. These tests were performed on a computer running Windows 7 on an Intel Core® i5-2410M 2.3GHz processor, with 6GB of RAM.

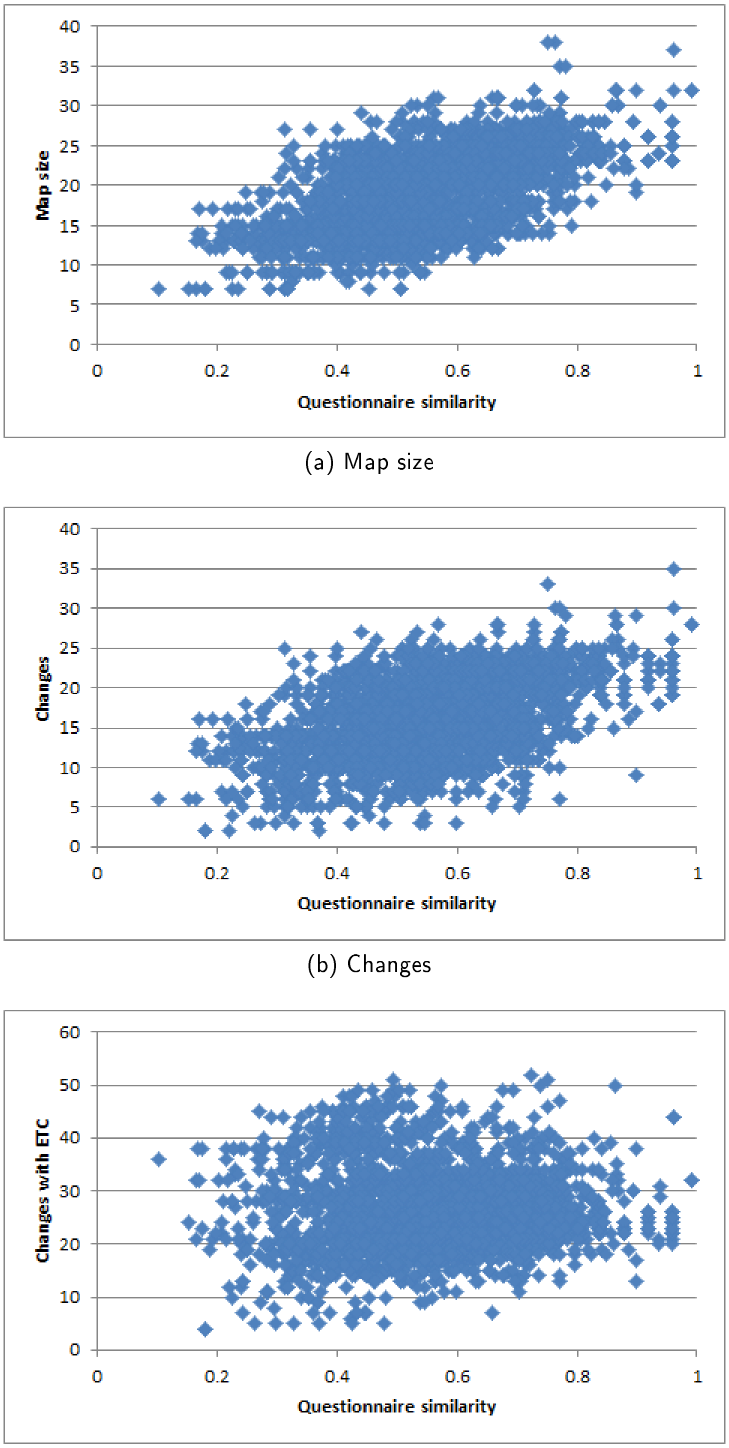
For operations 1 and 2, 11 iterations were performed, and the first iteration was discarded as it was consistently found to have a longer execution time, and this is attributed to implementation issues within Java.[6] For opening a file, the first iteration took typically 2-3 times longer than subsequent iterations. Operations 3, 4 and 5 require significantly more processing and so only two iterations were performed, and the first was discarded.

1 *Opening a file:* 0.0023 seconds. Given a Java File object[7], this is the mean time taken to create a Garment object in memory. This value was obtained by first building an array of File objects within the case base. Then, the array was iterated through. The time taken for the last 10 iterations were averaged and divided by the number of garments (50).

2 *Computing a questionnaire similarity:* 1.7 x 10^{-6} seconds (to 2 significant figures). All unique pairs of garments were computed, where the garment objects had previously been loaded into memory. The time taken for the last 10 iterations were averaged and divided by the number of pairs (2450).

[6] These could include, for example, the global caching of String objects.

[7] Java File objects hold no significant data other than the file name and path.



(a) Map size

(b) Changes

(c) With ETC

Figure 8.14: Scatter plots with Questionnaire similarity

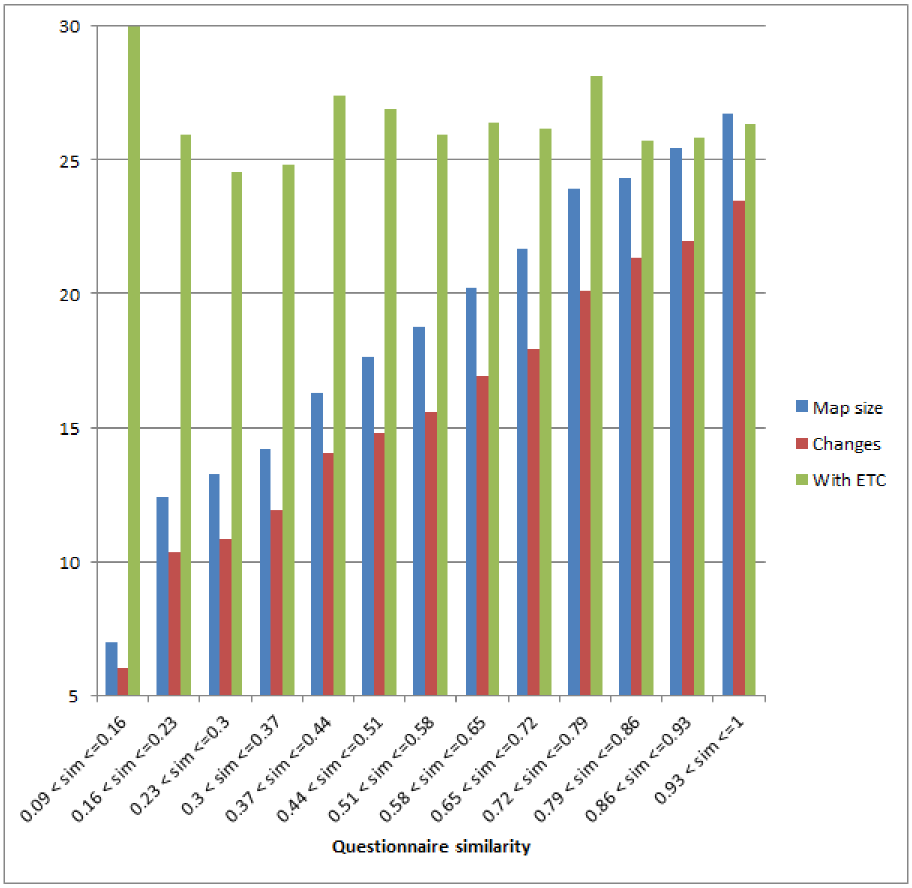


Figure 8.15: Questionnaire similarity histogram

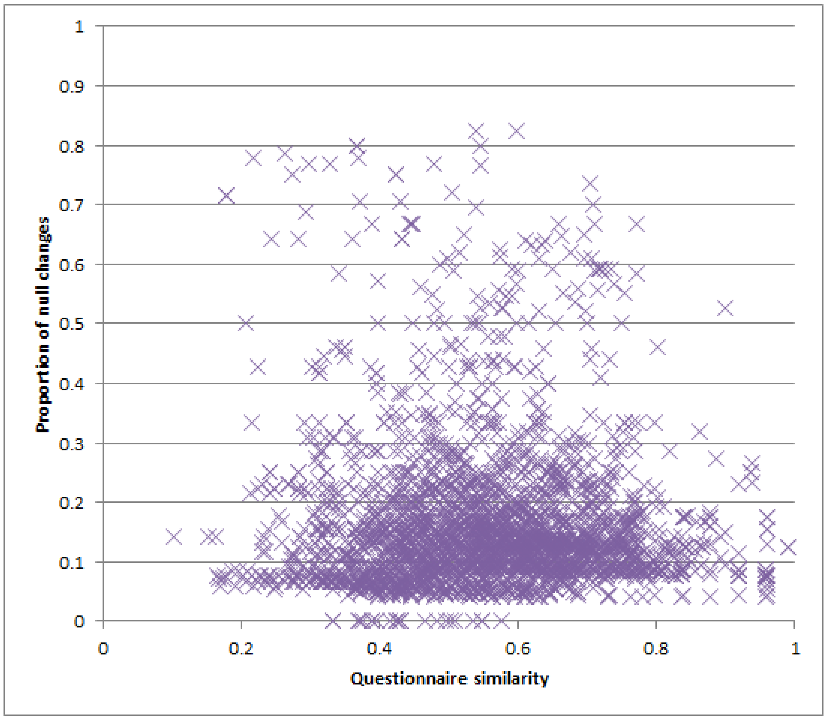


Figure 8.16: Proportion of null changes versus questionnaire similarity

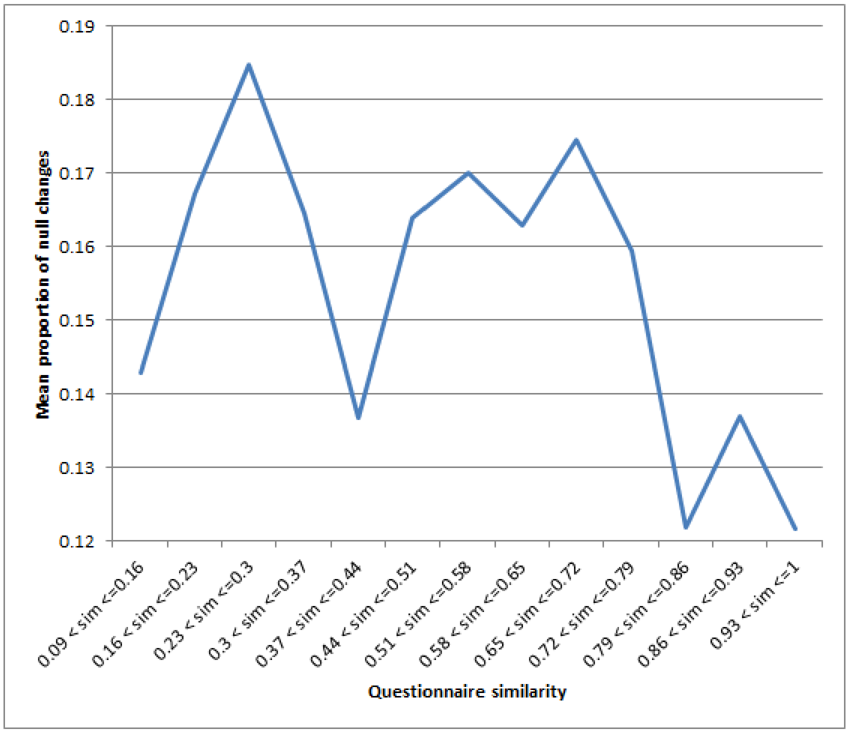


Figure 8.17: Mean proportion of null changes versus questionnaire similarity

3 *Determining the effects of adaptation:* 0.0073 seconds. Adaptation was attempted with all unique pairs of garments, where the garment objects had previously been loaded into memory. This operation determines the effects of adaptation, and whether it is technically successful or not; however, the changes are not actually made to the query case at this stage. The time taken for the last iteration was divided by the number of pairs (2450).

4 *Apply the effects of adaptation:* 0.0004 seconds. The methodology was as per test 3, except a record was formed of pairs of garments, where adaptation would be successful. The record included the list of changes that adaptation proposed. Then, the successful records were iterated over and the output of adaptation was obtained in volatile memory. The time taken for the last iteration was divided by the number of pairs for which adaptation was a technical success (1562).

5 *Save a file:* 0.0032 seconds. The methodology was as per the test 4, except an array of garments that formed the result of successful adaptations was compiled. The array was iterated over and each garment was saved using a unique file name. The time taken for the last iteration was divided by the number of pairs for which adaptation was a technical success (1562).

The tests show that the times taken for operations 2 and 4 are insignificant. However, these operations may have to be invoked for all cases in the case base. The time taken for operation 5 is much greater, but this is typically performed only once per query case. The combined duration of operations 1 and 3 is approximately 0.01 seconds.

8.4 Estimating the effect of case base size

As discussed previously, the experiments presented in section 8.3 were conducted with a case base of 50, which gives an effective size of 49 using the leave one out methodology. One of the assumptions made in section 8.1 is that performance would improve with case base size. Unfortunately, this assumption is impossible to test without more cases. However, it is an assumption that is normally shown to be correct in the literature (e.g. [173]), given that there is sufficient case base coverage and the cases are of good quality.

The assumption normally applies because CBR is normally used in domains in which similarity correlates with adaptability. In SEACOP, this implies that it is assumed that questionnaire similarity correlates with score. If the assumption holds, then it may be possible to extrapolate a relationship between case base size and score, based on the existing data about the distribution of questionnaire similarities. Using this extrapolation, an estimate of the number of cases required to yield good results (hypothesis 7) from SEACOP can be obtained.

In section 8.4.1, some results from the experiment (described earlier in this chapter) are presented, and then assumptions are stated and justified based on those results. Using those assumptions, a Monte Carlo simulation is performed, as explained in section 8.4.2.

8.4.1 Assumptions

8.4.1.1 Normally distributed questionnaire similarities

Figure 8.18 shows a histogram of the questionnaire similarities for all pairwise comparisons.

- Number of pairs: 2450
- Mean: 0.5560
- Median: 0.5546
- Standard deviation: 0.1477
- Kurtosis: -0.1299
- Adjusted Fisher-Pearson standardized moment coefficient (skewness): 0.0814

From the kurtosis value, it is apparent that the distribution of questionnaire similarities is approximately Gaussian. For the purposes of the simulation, it is assumed this distribution applies to the problem domain in general.

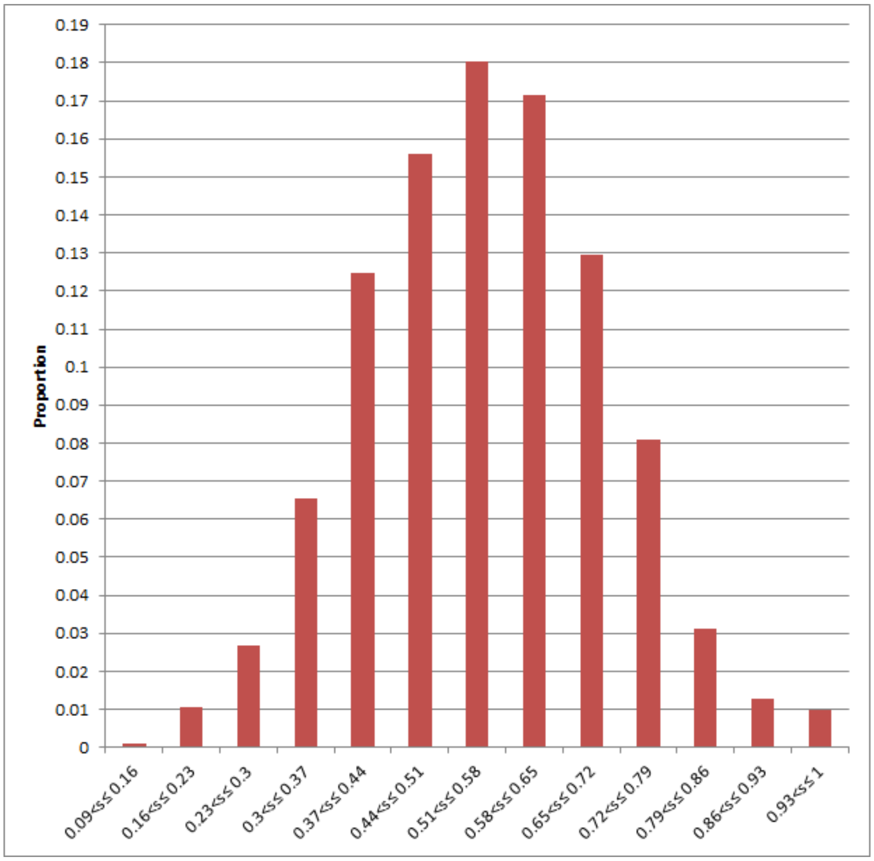


Figure 8.18: Distribution of questionnaire similarities

8.4.1.2 Relationship between questionnaire similarity and score

The relationship between questionnaire similarity and score was discussed in section 8.3.4.1. In this section, a function is sought which takes the questionnaire similarity as its input, and outputs an approximate value for the score. A regression analysis of score on questionnaire similarity was performed, but unfortunately all the models yielded a maximum score of 4 within the range (i.e. $0 \leq sim \leq 1$). Since the models ignored results with a score of 5, they were rejected.

Instead, an invertible function which takes $score$ values in the range 1 to 5 (inclusive) as inputs, was sought. This function should be monotonic, since this exercise is based on the assumption that there is a correlation between questionnaire similarity and score (and a non-monotonic function would not be invertible).

Figure 8.19 shows a regression analysis of questionnaire similarity on score, using a logarithmic function. Table 8.7 shows the different models that were used, the function within that model that was the best fit, and the respective error value. The logarithmic model was chosen, since it is monotonic, and has a higher R^2 value than the exponential, linear or power models. The polynomial functions were not used as they are not monotonic, even within the range required (i.e. $1 \leq score \leq 5$).

Table 8.8 and figure 8.20 show a comparison of the results of the logarithmic regression with the mean values for questionnaire similarity. The cases with a score of 5 have an estimated value (which is

Model	*Equation*	*Error R^2*
exponential	$sim = 0.6022e^{0.0989score}$	0.2958
linear	$sim = 0.0766sim + 0.5935$	0.3356
logarithmic	$sim = 0.2033\ln(score) + 0.615$	0.3633
power	$sim = 0.6183score^{0.2637}$	0.3235
polynomial	$sim = -0.0263score^2 + 0.2324score + 0.3933$	0.3902
	$sim = -0.0156score^3 + 0.1115score^2 - 0.1352score + 0.6814$	0.4216

Table 8.7: Regression analyses: questionnaire similarity versus score

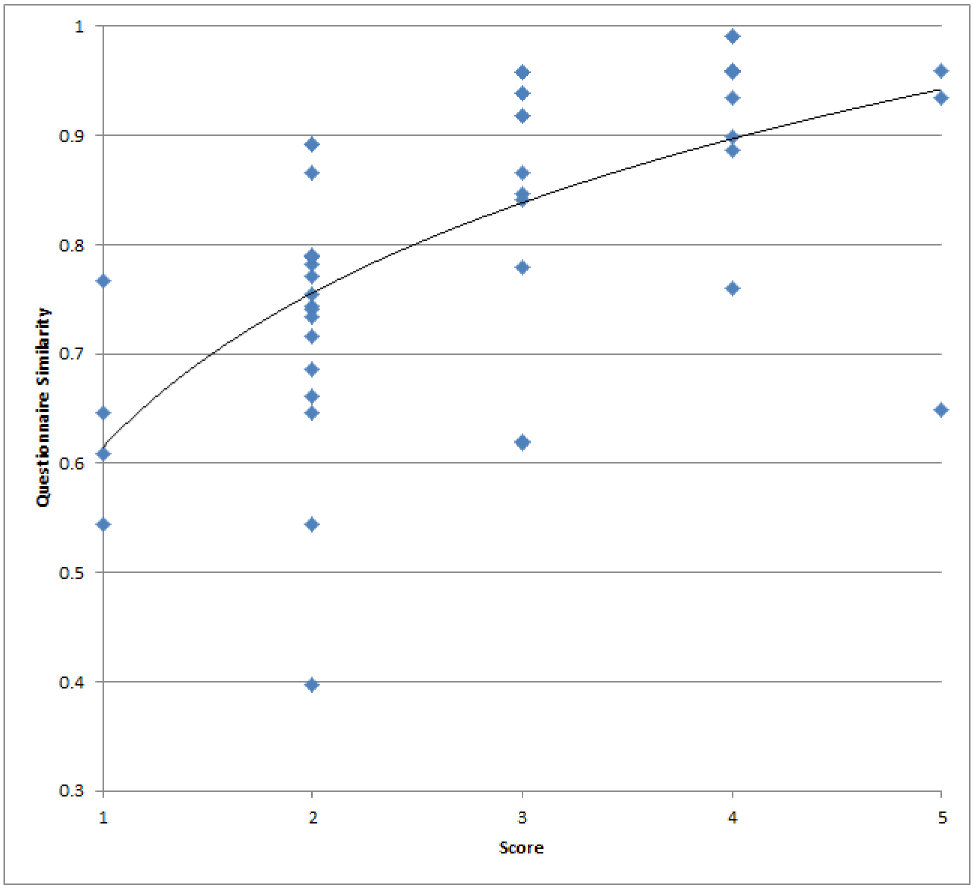


Figure 8.19: Questionnaire similarity versus score, showing logarithmic regression model

Score	*Actual*		*Estimate*	
	Mean	*Number of Values*	*Value*	*Deviation (2 d.p.)*
1	0.6416	4	0.6150	4.15%
2	0.7330	18	0.7559	3.13%
3	0.8583	13	0.8383	2.32%
4	0.9297	10	0.8968	3.54%
5	0.8477	3	0.9422	11.15%

Table 8.8: Comparison of estimated and actual values for questionnaire similarity, given the score

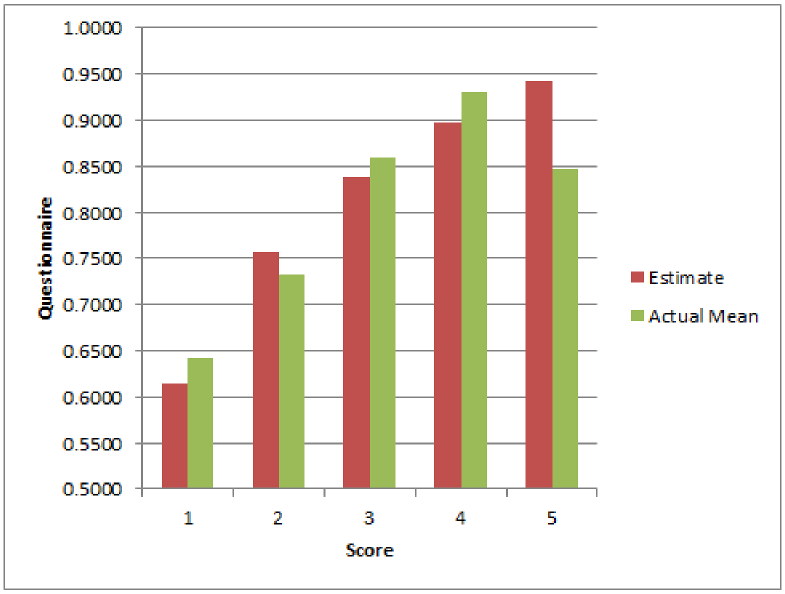


Figure 8.20: Comparison of estimated and actual values for questionnaire similarity, given the score

based on the logarithmic function given above), which deviates from the actual mean value by 11.15%. This deviation is fairly high, but the sample size is very small: there are only three cases. Also, the model performs well for the other cases, with the maximum deviation being 4.15%. The percentage deviation is simply calculated as:

$$\frac{|estimate - actual|}{actual} \times 100$$

The logarithmic function is deemed a good approximation, since it fits well (except for the 3 high scoring cases). The inverse of this function is:

$$score = e^{\frac{sim-0.615}{0.2033}}$$

This value is bounded and rounded to the nearest integer thus:

Questionnaire Similarity	*Technical Success*	*Technical Failure*	*Proportion Success*
0<s≤ 0.05	0	0	
0.05<s≤ 0.1	0	0	
0.1<s≤ 0.15	0	1	0
0.15<s≤ 0.2	5	7	0.4167
0.2<s≤ 0.25	24	9	0.7273
0.25<s≤ 0.3	35	13	0.7292
0.3<s≤ 0.35	78	28	0.7358
0.35<s≤ 0.4	104	62	0.6265
0.4<s≤ 0.45	156	89	0.6367
0.45<s≤ 0.5	172	96	0.6418
0.5<s≤ 0.55	193	119	0.6186
0.55<s≤ 0.6	200	111	0.6431
0.6<s≤ 0.65	190	112	0.6291
0.65<s≤ 0.7	153	87	0.6375
0.7<s≤ 0.75	107	66	0.6185
0.75<s≤ 0.8	70	47	0.5983
0.8<s≤ 0.85	35	22	0.6140
0.85<s≤ 0.9	16	9	0.6400
0.9<s≤ 0.95	10	4	0.7143
0.95<s≤ 1	14	6	0.7000
0.95≤s≤ 1	1562	888	0.6376

Table 8.9: Proportion of technical successes versus questionnaire similarity

$$integerscore = \begin{cases} 1 & if\ score < 1.5 \\ 5 & if\ score \geq 4.5 \\ round(score) & otherwise \end{cases}$$

The effect of the rounding function is to create boundaries with approximate values {0.6974, 0.8013, 0.8697, 0.9208}. These boundaries delineate 5 intervals which correspond to the scores 1 to 5, respectively.

8.4.1.3 Constant probability of technical success

Table 8.9 and figure 8.21 show the proportion of adaptations which were technically successful, for all pairs of cases in the case base, grouped by questionnaire similarity. Technical success means that an output was produced without violating the fundamental constraints (crossovers and non-intersects), being the opposite of technical failure, when no output was produced. However, technical success does not necessarily mean that the output was good.

Figure 8.21 shows that, apart from the 7 pairs with a similarity below 0.2, the proportion of technical successes lies in the range 0.5983 to 0.7358. The overall mean is 0.6376. As discussed in section 8.4.1.1, the distribution of questionnaire similarities is approximately Gaussian. This means that the high and low similarity bands, which have a proportion that is visibly different from the mean, represent very few cases.

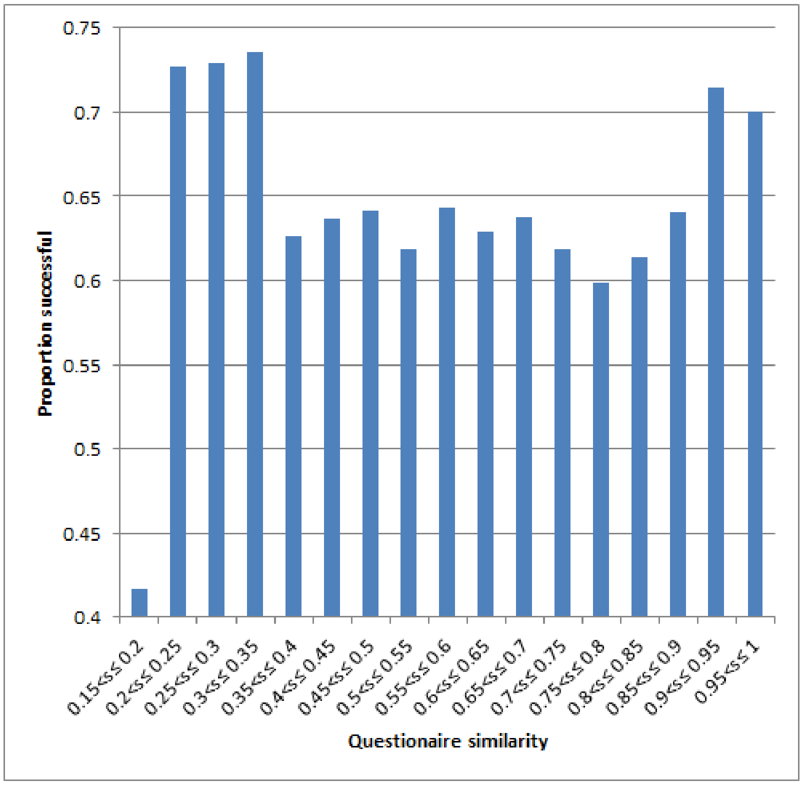


Figure 8.21: Proportion of technical successes versus questionnaire similarity

The histogram does not indicate that there is an obvious correlation between the two factors and it seems reasonable to assume a constant probability of technical success of 0.6376 (the value of the overall mean from table 8.21).

8.4.2 Monte Carlo simulation

Monte Carlo simulations use the repetition of stochastic techniques to make generalisations about the long-run outcome of events [166]. The object of the Monte Carlo simulations discussed here was to estimate the effect of the case base size (n) on the score. Each run of the simulation worked by generating an array of n numbers with a Gaussian distribution as per section 8.4.1.1, to simulate the questionnaire distances. These numbers were then sorted in descending order.

The process iterated through the array (starting with the maximum value) in a way that simulated retrieval. Upon each iteration, a pseudo-random number with uniform distribution was generated; if this exceeded a preset value (0.6376, as per section 8.4.1.3) then adaptation was deemed to be successful. Otherwise, the next largest value from the array was obtained and the process repeated, until adaptation was deemed successful or all the values in the array have been "retrieved". The latter situation represented technical failure.

If adaptation was deemed technically successful, then the similarity was converted into an integer score as per the formula at the end of section 8.4.1.2. The results were collated over 10000 runs of the

Case base size	*Failure*	*Proportion with score*				
		1	*2*	*3*	*4*	*5*
25	0	0.0580	0.4012	0.3035	0.1338	0.1036
50	0	0.0032	0.2052	0.3762	0.2219	0.1935
75	0	0.0002	0.0947	0.3489	0.2793	0.2769
100	0	0.0000	0.0445	0.2948	0.3085	0.3523
125	0	0.0000	0.0196	0.2387	0.3229	0.4188
150	0	0.0000	0.0090	0.1896	0.3248	0.4766
175	0	0.0000	0.0041	0.1496	0.3181	0.5283
200	0	0.0000	0.0019	0.1131	0.3069	0.5782
225	0	0.0000	0.0008	0.0884	0.2926	0.6183
250	0	0.0000	0.0004	0.0666	0.2719	0.6611
275	0	0.0000	0.0001	0.0511	0.2511	0.6977
300	0	0.0000	0.0001	0.0385	0.2319	0.7296
325	0	0.0000	0.0000	0.0304	0.2170	0.7525
350	0	0.0000	0.0000	0.0224	0.1980	0.7796
375	0	0.0000	0.0000	0.0179	0.1809	0.8012
400	0	0.0000	0.0000	0.0140	0.1651	0.8209
425	0	0.0000	0.0000	0.0105	0.1479	0.8416
450	0	0.0000	0.0000	0.0075	0.1358	0.8566
475	0	0.0000	0.0000	0.0054	0.1217	0.8729
500	0	0.0000	0.0000	0.0045	0.1113	0.8842
49	0	0.0035	0.2118	0.3737	0.2195	0.1915

Table 8.10: Monte Carlo Simulation results

experiments; they are shown in table 8.10. The results indicate that if the case bases is larger than around 175 cases, the system can be expected to give an excellent result (score of 5) the majority of the time.

An investigation was performed to determine the effect of the probability of technical success on case base size which is required to achieve 'excellence'. Excellence is defined as the state in which at the score is 5 for at least half the query cases. This means that a very good score is being obtained the majority of the time.

The investigation took the form of a search. For a given case base size (n), the simulation was run 1000 times for n=100 cases, and then n was increased by 100 each time (if necessary) until excellence was achieved. Then a second search was conducted with n starting at the output of the previous search, and using runs of 10000 simulations and changing the case base size in (either positive or negative) increments of 10. Finally, there was a third search which incremented or decremented n in steps of 1, using runs of 100000 simulations. The lowest value of n from the third search which achieved excellence was output.

The results are shown in figure 8.22. The relationship is clearly a reciprocal or inverse power one. This indicates that if there is a small error in the probability of success, there is only likely to be a small error in the cases required for excellence, as long as the probability of success is not too low (e.g. if it is > 0.4).

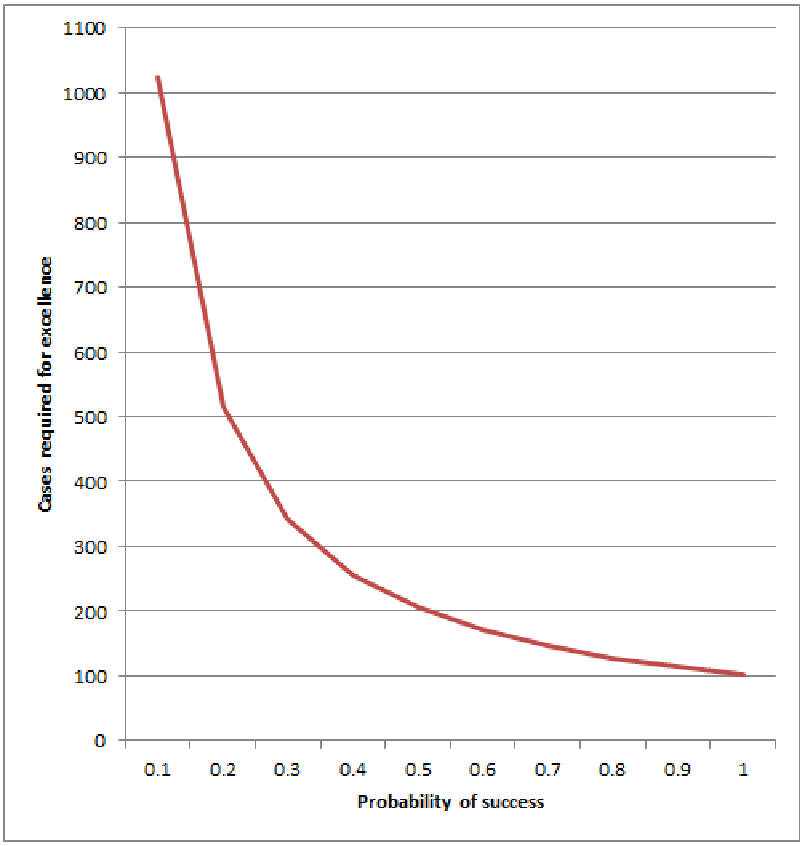


Figure 8.22: Effect of the probability of success on the cases required for 'excellence'

8.5 Discussion

This section discusses the significance of the results of the experiment and simulation. Care has been taken in the interpretation of the results for a number of reasons. For example, non-uniform distributions affect the analysis; when Gaussian data is grouped in a histogram this gives a different correlation to the use of the raw data. And even if a correlation is present, this does not necessarily imply causation. Also, sample sizes have to be taken into consideration: smaller samples mean that trends or relationships may be less obvious or those that are apparent may actually be spurious.

The domain of knitting tends to lend itself more to heuristics rather than formal logic, and this chapter contains no proofs.[8] However, there is substantial evidence that hypothesis 1, 2, 4, 5, 7 and 8 are correct. There is some evidence to support hypothesis 3 (use of MAD as an indication of quality). There is substantial evidence that hypothesis 6 is incorrect, as explained in section 8.5.1.

Algorithms for implementing similarity and adaptation were described in section 6.2 and chapter 7. However, no coherent strategy for implementing these and integrating them with the computer aided design (discussed in chapter 5) has been presented so far. Such a strategy can only be stated with confidence when it is accompanied by evidence of its efficacy; such evidence is discussed in section 8.5.2.

8.5.1 Significance of the results

As discussed in section 8.3.1, the 'all move advisers' mode gave the best results, so this provides clear evidence that hypothesis 2 is correct. In the alternative mode, the only way that move advisers are involved in RDR is indirectly, via the retrieved case.[9] For example, if both retrieved and query cases have a round neck, then the action of making that neck deeper will be a part of the RDR. When the neck was made deeper in the retrieved case, presumably a move adviser would have been invoked to move the Bézier curve control points, and thus the action of that move adviser will then be replayed.

However, if the retrieved case had a V neck and the query case a round neck, then the neck deepening will be replayed, but there would be no Bézier curve control points in the mapping, as these are not present in the retrieved case. Therefore, the character of the curve is in danger of being distorted by RDR. This phenomenon is presumably responsible for the improved results when all move advisers are being used (as per hypothesis 2). It is better to involve the move advisers that are known to be appropriate to the query case (as per the example in section 8.3.2.1), than try and use move advisers directly from the retrieved case, which may have different features.

Hypothesis 3 was that MAD could provide an indication of quality. The results in section 8.3.3 show a negative correlation between MAD and score. Additionally, as the scores increase, the range of MAD

[8] Although, proving a hypothesis is notoriously difficult since it is necessary to disprove any alternative causes of the phenomenon.

[9] With the exception of the level 2 move advisers which maintain the integrity of the sketch.

values decreases. This is to be expected, as it indicates that excellent results have a very low MAD. Evidence suggests that MAD provides an indication of quality, but is not synonymous with the score. As discussed in section 8.2.3, MAD and the score measure different things.

The most appropriate use for MAD might be to validate the score in future experiments. After the score has been input by reviewers, this could be checked against the MAD. An example of a function that may help to implement this was shown in section 8.3.3. In the case base shown, 12% of user-provided scores deviated from the estimated score by 2 or more points. In these scenarios, a warning could be flagged to prompt the user to re-evaluate the result, or pass it to another reviewer for an extra opinion.

Hypothesis 1 was that the use of CBR should outperform rules. The results in section 8.3.1 suggest that this would not be true if CBR was used exclusively, in a small case base. In this scenario, rules produced outputs with the best mean score. However, for some cases the CBR output was better, so it is possible that the designer could be offered both alternatives, and invited to make a choice. The combination of rules and CBR would then be better than rules alone (by definition). This is discussed further in section 8.5.2.

The results in section 8.3.4.1 suggest a positive correlation between questionnaire similarity and score, confirming hypothesis 4 that the effectiveness of CBR decreases with increased distance. There are outlier cases which indicates that the questionnaire similarity is not a perfect measure, but this is to be expected.

Section 8.3.5.1 contained some thought-provoking results about the relationship between the questionnaire similarity and the amount of change. It is clear that the number of changes before ETC correlates with questionnaire similarity. This result appears odd because an intuitive response would be that the more similar two objects are, the less change is required to adapt one into the other (as per hypothesis 6).

However, it is also intuitive that the map size should correlate with questionnaire similarity, since map entries can only be added if features are in common to the two cases. Changes (before ETC) can only occur because of map entries; this means that when two garments are similar to each other, there are more features in common and therefore more opportunity to change one into the other. Conversely, two dissimilar garments will have less features in common and there is less opportunity for change. This is strong evidence for the contrary of hypothesis 6: in fact as questionnaire similarity increases, the amount of change in adaptation will *increase*.

This phenomenon is the most likely explanation for the high rank (i.e. low similarity) of the retrieved case for s3041, as presented in section 8.3.2.3. The sketch instantiation algorithm guarantees that the *rules output case* will not contain any violations of the fundamental constraints. This could explain why adaptation was successful with such a dissimilar garment: because there were fewer (18) changes, there were less opportunities for the *rules output case* to be made invalid. The situation in figure 8.8, where adaptation failed, involved 33 changes.

A negative correlation between similarity and the probability of technical success seemingly contradicts one of the established principles of CBR: that similarity has a positive correlation with adaptability. It is important to recognise, however, that technical success is different from the score, which tends to increase with increasing similarity.

8.5.2 Recommendations for Use

The results in section 8.3.4.2 suggest that CBR tends to perform better than the rules where the questionnaire similarity exceeds a threshold of 0.93. In fact, the rules were never better (than CBR) above this threshold. This is good evidence to support hypothesis 5: the threshold on similarity could be used as the basis for an automated decision about whether to invoke CBR or not.

The recommended strategy for using the similarity and adaptation algorithms, supported by the experimental results, is as follows:

1 The user enters the query case (Q), which consists only of a questionnaire.

2 The similarities of all existing cases with the query case are computed.

3 The set of cases (s) with a similarity value of 0.93 or greater is formed.

4 If (s) is empty, the result of the rules is output and the process moves to step 10.

5 The most similar case (R) is removed from (s).

6 Adaptation is attempted with (Q) and (R).

7 If adaptation is unsuccessful, the algorithm moves back to step 4. Otherwise, it moves to step 8.

8 The user is asked whether they accept the case. If they do not, the process moves to step 4. Otherwise, it moves to step 9.

9 The result of CBR is output to the user.

10 If necessary, the user edits the output.

11 The output is retained as a new case.

The adaptation process is fully automated, but is only attempted with similarities over the threshold. This means that the likelihood that the output of adaptation will be acceptable to the user is high. If the threshold is not met, then the output of the rules is used as the evidence in section 8.3.4.2 suggests this is likely to be better. In the case base of 50 garments, 16 had at least one other case with questionnaire similarity above the threshold. However, as the case base increased in size, this proportion would increase.

This threshold should be subject to review, but my expectation is that it will remain in the range 0.8 to 0.95.

The simulation in section 8.4.2 assumed the system would always use the output of CBR. It also made several other major assumptions, as detailed in section 8.4.1. Because of these assumptions, the results of the simulation must be interpreted liberally, rather than literally.

The absence of failure is notable in table 8.10. However, it is easily explained by the assumption in section 8.4.1.3, and the methodology of the simulation which treats each instance of adaptation (for the same query case) as statistically independent.[10] For 25 cases the probability of adaptation failure is assumed to be $(1 - 0.6376)^{25} \simeq 10^{-11}$, and for 500 cases it is $\simeq 10^{-221}$. The assmption of statistical independence is naive, since some of the factors that lead to adaptation failure are dependent only on the nature of the query case (as explained in section 8.3.2.3), and therefore the events are not independent. A more sophisticated version of the simulation (which was not implemented) might assume that the probability of technical success varies from query case to query case, using pseudo-random numbers.

However, it is worth noting that the assumption about the relationship between questionnaire similarity and score in section 8.4.1.2 is pessimistic. For scores of 5, the regression model gives a questionnaire similarity that is higher than the mean of the three values. The simulation suggests that excellent results will be obtained with a case base of 175 cases. However, even if this figure lies in the range 100-300 cases, this would still mean that CBR should significantly outperform the rules (for the production of sketches) within a year in an organisation such as Sirdar Spinning Ltd, which produces approximately 300 patterns annually.

The decision on whether to accept the output of CBR must lie with the user since there will always be outlier cases where CBR performs poorly, even with a high questionnaire similarity. Through the use of the threshold, the situation where the user is presented with a succession of poor cases is likely to be rare. Such a situation could to disillusion the user with CBR and induce them to reach for the safety of the rules.

The study on efficiency in section 8.3.6 indicates that the time taken to compute questionnaire similarity is negligible. In a hypothetical scenario where the case base has 1000 garments and it is necessary to open and check adaptation for all of them with the same query case, this would be expected to take of the order of 10 seconds. Whilst 10 seconds may be an acceptable time frame, the scenario presented is pessimistic. As discussed in section 8.5, realistically a lot less processing than this would be performed, as the strategy outlined above restricts attempts at adaptation to cases that have a similarity exceeding the threshold. Through the use of this strategy, it is likely that case base maintenance would

[10] False assumptions of statistical independence are sometimes widely accepted and lead to erroneous conclusions. For example, Professor Sir Roy Meadow famously stated that the chances of 3 babies dying naturally were 1 in 73000000, a figure obtained by cubing the probability of one cot death (1 in 8543). However, the events were not necessarily independent and the murder conviction (in which this evidence was cited) was overturned on appeal [176].

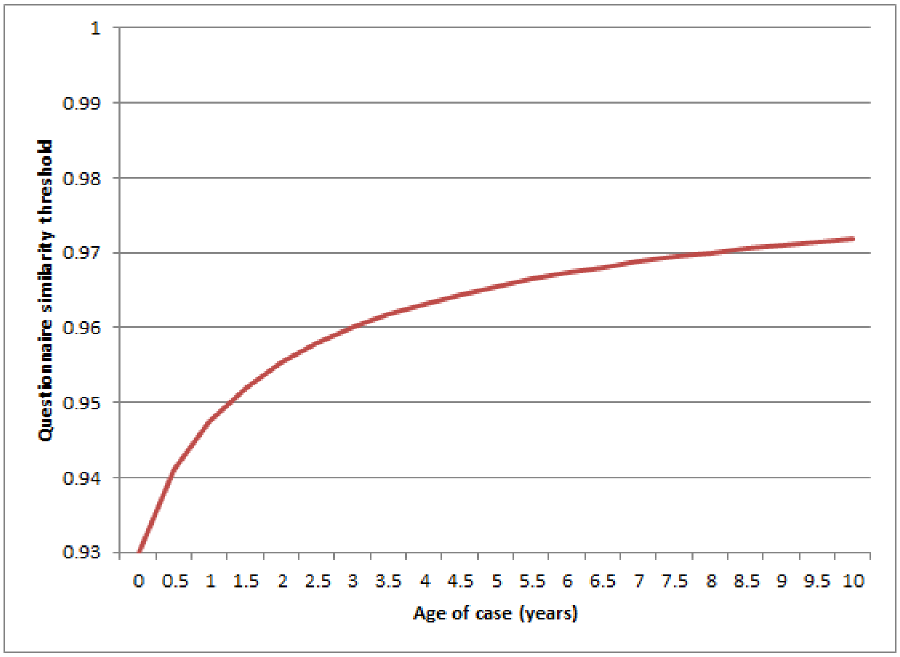


Figure 8.23: Illustration of function which increases threshold with age

not be required for reasons of efficiency alone, even if the case base contained thousands of garments.

Effective retrieval can be guided by factors apart from similarity. Newer garments are more relevant than older garments, as design trends change with fashion. So, it may prove beneficial to alter the threshold on questionnaire similarity according to the age of the retrieved garment. Figure 8.23 gives an example of such a function.[11] Design departments typically want to retain a record of all their designs, for operational purposes (e.g. they may be required to offer customer support on old designs). Through the use of a variable threshold, the system could store designs indefinitely, whilst maintaining good performance.

The results and analysis in this chapter have shown how RDR fits in to the general framework of CBR. Some aspects of the results agree with those found in the CBR literature, for example similarity being a guide to retrieval. Others are surprising, such as the amount of change increasing with similarity. The results also show that, at least in this domain, RDR is dependent on the supporting process for dealing with outlier cases and as an adjunct to CBR when the case base is small.

It has been shown that, for very similar cases, the use of RDR can outperform the rules alone. This has the potential advantage of saving labour, i.e. allowing designs to be produced more quickly. However, RDR has one disadvantage that has not yet been discussed. The user carries a heavy burden with SEACOP, since they are responsible for ensuring the quality of the output. Some users may prefer the predictability of rules; the rules give them a sketch which is usually inaccurate but forms a consistent starting point. In contrast, CBR provides a sketch with an inconsistent starting point which is, for large case bases, often better than the rules.

[11] The function used in this example is $threshold = 0.99 - \frac{0.06}{\sqrt{age+1}}$.

Chapter 9

Conclusions

> "Some problems are so complex that you have to be highly intelligent and well informed just to be undecided about them." - Laurence J. Peter

The main challenges in this PhD were the subjective nature of success, design constraints, the complexity of the objects, and the lack of readily available and usable data as examples. It has been demonstrated that a good solution to the problem can be obtained with a hybrid CAD-CBR system.

9.1 Main contributions

This research made a number of contributions to knowledge, and satisfied the objectives listed in the introduction chapter (section 1.4).

A CAD system which meets many user requirements and also facilitates CBR

One practical outcome of this PhD dissertation is a CAD system that automates the knitwear design process. The system responds to the user's requirements, as expressed in the questionnaire. The system also supports automatic enforcement of constraints such as sleeve-arm consistency, so the designer can be assured their work constitutes a knittable garment. This thesis documents the prioritisation scheme that is used to enforce and fix constraints in the sketches.

Rather than utilising an existing CAD package, the decision was taken to develop a bespoke system. Whilst this has obvious drawbacks in development time, it has many advantages. With a third party system, access may be restricted to only the data files, or perhaps to selected volatile data, via an API. However with SEACOP, the full object model and program code is available for use in experiments and in developing a hybrid CBR system. This ensures that the more difficult aspects of the implementation, such as the implementation of the rules about mapping Shape Points, were not fettered by limitations of access to data.

There was a strong symbiosis between CAD and CBR, both in the functionality and the program code that was used to implement them. The concepts of *move advisers* and *exploring the consequences* are used for both managing constraints in the sketch editor, and for implementing changes in adaptation. The sketch instantiation algorithm is able to produce a consistent starting point for either user editing or CBR. The CAD functionality supports CBR in two additional ways: it helps with initial seeding of the case base, and it provides a fall-back for outlier cases, when CBR gives a poor result. A strategy for combining CAD and CBR was proposed, and evidence of its efficacy presented; this fulfils objective (i) of section 1.4.

The use of a 2-level structure as a design specification, and self-index for CBR

SEACOP represents designs on three levels: questionnaire, sketch and chart. The specification is dual-purpose. As its name suggests, it specifies the design, and therefore provides a means of generating the sketch via the rules in the sketch instantiation algorithm. It consists of sufficiently few questions to be completed relatively rapidly by a user. However, there is enough information in the specification to generate a sketch which provides an approximate solution to the user's requirement.

More importantly, the questionnaire provides a means of implementing a similarity measure for CBR. The measure is efficient to execute, taking of the order of a microsecond to compare two cases. It is a data structure that is inherent to the problem, and avoids the need for an artificially constructed index. An algorithm for comparing questionnaires was presented; this fulfils objective (ii) of section 1.4.

The use of ranked attributes for similarity

A simple algorithm and accompanying GUI has been constructed; this is capable of generating weights from a list of features that are ranked in order of importance. Many domain experts find the idea of providing weights difficult, but it is much easier to rank features in order of importance. A choice of several functions was provided to map ranks to weights; this affords flexibility. The algorithm is also flexible enough to deal with irrelevant attributes and those that are of equally high priority. It has been shown how the global similarity measure can be used with different local similarity measures, e.g. a Likert matrix.

Rule difference replay: a novel generative adaptation operator

A particularly significant outcome of this research has been the development of a novel generative adaptation operator, *rule difference replay* (RDR). It has been demonstrated that RDR can produce positive results in the knitwear problem. The assumptions that underpin RDR have been clearly stated and justified, so that the wider relevance of the technique (beyond the knitwear problem) can be established.

It has been shown that RDR is capable of correcting deficiencies in rules, where those deficiencies have previously been corrected manually.

RDR can be implemented without an onerous knowledge engineering task. Therefore, in situations where it is applicable, it may be an attractive alternative to knowledge-intensive adaptation. The biggest burden in RDR is likely to be establishing common features between the objects. However, if a system is constructed with the requirements of RDR in mind from the start, this may not be too difficult. The way in which the burden was overcome within SEACOP (by labelling points in the designs with roles) has been described and demonstrated. RDR enables the solutions to problems to be reused, thus fulfilling objective (iii) of section 1.4.

Minimum average distance: a measure of solution similarity

Minimum average distance (MAD) was devised: it is simple algorithm for comparing shapes, drawings or sketches. MAD is dependent only on the ability to derive common features between the shapes, and makes no other assumptions apart from there being only one global minimum.

It has been shown how MAD can be used to validate subjective success scores, so that more confidence can be attributed to those scores. The way in which deviations between actual and predicted scores are highlighted has been demonstrated. It has also been shown how the combination of subjective scoring and validation with MAD allows success to be measured; thus objective (iv) of section 1.4 has been fulfilled.

When similarity does not correlate with adaptability

An anomalous situation in which similarity does not correlate to adaptability was encountered, and explained. The situation can arise where an object starts in a valid state, where that starting point is very close to an invalid state (i.e. it is a very sensitive solution), where adaptation can potentially make it invalid, and when only features that are in in common to the query and retrieved cases are changed. This can result in a negative correlation between similarity and adaptability. This phenomenon could occur in other domains with RDR; however this could be remedied, by using similarity as a guide to retrieval. The situation is noteworthy because it is a finding of this research that apparently contradicts with one of the established assumptions of CBR: that similarity has a positive correlation with adaptability.

A simulation for estimating the effect of case base size

A simulation which estimates the effect of case base size on the performance of the system was presented. The simulation is useful in problems such as the knitwear one, where access to cases is limited. The results indicated that a relatively modest increase in case base size is likely to mean that the CBR part of SEACOP yields excellent results in the majority of cases. This answers the question posed in objective (v) of section

1.4.

9.2 Applicability to other domains

Much of the functionality developed as part of this thesis could be reused for other purposes, for example the computer-aided design tool could be adapted for other 2-D design problems. The concept of comparing specifications in order to measure the similarity of domains could also be extended to other domains.

The most promising opportunities to extend this work to domains arise from RDR. The assumptions on which RDR relies on have been carefully documented in this thesis. RDR has the potential to be reused in design problems as architecture or interior design.

RDR could also be applied to other domains, for example metal cutting. In order to meet the assumptions, there would need to be common features between the objects being manufactured. However, it is easy to see how this could be achieved if the objects came from the same family of products, e.g. if they were all types of spanner or angle bracket.

There is no reason why RDR should be restricted to 2D-CAD; differences along three dimensions could be replayed. Also, RDR could potentially be used with commercial CAD systems, provided sufficient access to meta-data about the drawings (e.g. the function of objects and of parts of those objects) is available.

Interior design is applied to domestic, commercial or industrial environments. Similarly to knitting, it is a weak theory domain it is hard to formulate rules about what constitute a good design, but most people know a bad design when they see it. One aspect of interior design that is particularly amenable to RDR is the placement of objects within a room. The objects could be decorative (e.g. ornaments or indoor plants), or functional (such as chairs or commercial office furniture). The representation of these objects in a computer could take several forms: as a pictorial image, but also as dimensions, and codes to indicate their function.

Interior designs could be created using simple domain rules. There is also the possibility to incorporate interesting variations on these rules, e.g. feng shui. But the rules are likely to be incomplete, or produce outputs that may be considered naive in some situations. So, the difference between the output of the rules and the resulting design could be replayed with RDR. The replay could take the form of alterations in the positions of objects, or changes to their dimensions or size. As with the knitwear design problem, this has the potential to be able to make up for shortcomings in the rules. For example, the rules may place the same size of desk in every office, but RDR may make the desk bigger in an executive office, if this change was previously made manually to a design in the retrieved case.

9.3 Further work

Computer-aided design

Experience has shown that computer-aided design can take considerably longer to implement than CBR. In fact, it has been said that some commercial knitting systems have had of the order of 150 person-years of development time devoted to them [151]. Since perhaps only 2% of this time has been spent on SEACOP's CAD functionality, there are obviously many improvements that could be made.

However, many of these improvements would add little scientific value to the research. For example, the CAD functionality could be extended to produce other types of garments such as hats, gloves and scarves. These garment types are rarer than cardigans and sweaters, and so it may be that attempts to use CBR with the new designs are fruitless.

Grading is one area in which improvements could be fruitful. Only one way to fit knitting patterns into a shape was implemented; in reality there are several alternative options. If CBR was extended to include knitting charts, this could become more relevant, since for adaptation to work, the retrieved case is often resized to match the query case.

Similarity

Various experiments could be made to try variations on the way questionnaires are compared. For example, the experiments could be repeated with different options chosen for the function that converts ranks to weights. Also, it may be possible to use machine learning techniques to obtain the weights, as discussed in section 2.2.2.2. Finally, the way in which the weights are aggregated could be changed, for example using the Manhattan or cosine angle distance.

Case merging

Case merging has been discussed previously in this thesis, but not implemented as it was felt that the concepts in RDR needed to be verified first with retrieval of intact cases. Case merging has the advantage that smaller data structures are being retrieved, this means that there is a smaller search space and the mean similarities of the individual components tend to be greater than those of the object as a whole.

The first step in implementing case merging would be developing similarity measures which apply to individual pieces. Then, a retrieval algorithm and method of repairing incompatibilities between pieces would need to be formulated. Some suggestions for this are given in [14].

Adaptation Failure

When checking for violations of the fundamental constraints in the sketches, the algorithms test whether line segments, Bézier curves, and elliptical quadrants intersect with each other. Some tolerance is allowed in these tests, and it may be profitable to experiment with the threshold values on these tolerances. If the tolerances were increased so that the sketch was still valid, then adaptation would succeed more often, which could yield better results. Implementation of this would be non-trivial since there are several algorithms that would need to be changed, and not all of these currently use a threshold, i.e. it is not a simple matter of changing a literal in a program.

In this work the results of adaptation failure were not repaired, as the sketch violation repair algorithms can produce results which are unexpected. However, it is also possible that they could make small changes to an otherwise invalid sketch, to produce a valid output from adaptation. The algorithms could be invoked with a caveat: if the total amount of change during the repair operation exceeds a preset threshold, the repair is not invoked and adaptation is deemed to have failed. The threshold would need to be determined empirically.

Sketch Adaptation

The use of move advisers has been shown to be beneficial during adaptation. However, it may be that conditions should be placed on this, i.e. selected move advisers could be disabled during adaptation.

The adaptation rules have been drawn up using heuristics, but their efficacy has not been tested in detail. It may be desirable to test adaptation with various combinations of the rules being disabled. It is not practical to test all all combinations, but the relevance of the rules to the test cases can be used as a guide.

Similarly, the conditions on mapping could be experimented with, either by being made stricter or relaxed. If the conditions are too strict, very little mapping will occur, and the output of RDR will be similar or identical to that of the rules. Alternatively, if the conditions are too relaxed then there will be many changes as a result of RDR, but the danger is that these are detrimental changes and the rules output will be better than the RDR output. Experimentation can determine if the current conditions are optimal.

It may also be fruitful to explore mixing RDR with knowledge-intensive adaptation rules. Conflicts between the two would have to be managed, but it is possible that the latter brings greater accuracy to adaptation, albeit with a greater knowledge engineering cost.

Experimental methods

The experiments are performed on a small case base and involve subjective scoring. To increase the case base, commercially available patterns can be 'reverse compiled' to yield sketches. To enter another 50 cases would be expected to take of the order of 100-150 hours. Having a case base of 100 cases would give more confidence in the results of the experiments. It would also validate the results of the Monte-Carlo simulation.

Of course, success in CBR is about more than just the size of the case base; it needs to have good coverage of the problem space. The coverage does not need to be uniform, but representative of the sort of cases that are presented to the system, with scope to deal with outliers. Ideally, some analysis of the coverage would be performed, to determine whether there are any areas that need to be strengthened with more cases.

Ideally, the results of SEACOP would be evaluated by a group of trained users. If 5-10 volunteers could be found, they could be given approximately 3-4 hours of training (as a group) on the user interface, and then left to evaluate the results individually using the five point scale. Depending on the skill level of the volunteers, the whole exercise could be completed in a day. Access to volunteers would be the most difficult aspect; ideal candidates would be design students or those studying for degrees in knitwear. The mean of the individual values could be used as the score. Using less than 5 people would incur the risk that the training might have to be repeated if users withdrew from the programme, since I believe at least three scores would be required per case.

Some improvements could be made to the MAD measure of solution similarity. Firstly, to improve confidence in the algorithm an in-depth investigation into the issue of local minima could be undertaken. Perhaps a mathematical proof could be formulated that there is only ever one global minimum. Alternatively, if some shapes do have local minima, then the nature of these shapes could to be established, so that modifications can be made to the search strategy.

The use of a penalty in MAD to take into account unmatched Shape Points could be attempted. This would circumvent the issue that MAD only focusses on features in common, so that it ignores things such as the addition of a yoke.

As suggested previously in this thesis, MAD could be used to validate the subjective scores. This may be even more relevant if these scores were provided by external reviewers.

Simulation

The idea of varying the threshold on questionnaire similarity that decides if CBR is used with the age of the retrieved garment was suggested. This could be incorporated into the Monte-Carlo simulation. If it is assumed that the number of cases added to the system each year is constant, the effect of this parameter on the predicted success of CBR could be investigated. The effect of this is that it is likely that if only a small number (e.g. 25) of cases is added every year, the probability of retrieving garments that are sufficiently recent or have a sufficiently high similarity will become very low. Thus, the variable threshold should not be used in situations where very few cases are added annually; an improved simulation could quantify this.

Chart Adaptation

To maximise the real-world relevance of SEACOP, ideally adaptation should include knitting charts. The functionality to create charts has already been developed, and this could be extended to work with CBR. It may be possible to formulate a method of adaptation that could be described as chart difference replay. The edits that were made to the knitting chart that was produced from the rules could be applied to a new chart.

Chart difference replay is likely to be more difficult to implement than RDR, because charts have more complex constraints than sketches. As an alternative, some form of substitution or transformation may be appropriate. For example, the type of edging stitch or the method of grading could be reused.

9.4 Dissemination

Part of the work detailed in this thesis has been published in three papers: two conference papers and a journal article.

[152] "Automating a Knitwear Design Process Using Case-Based Reasoning"

Proceedings of the 10th International Conference on The Modern Information Technology in the Innovation Processes of the Industrial Enterprises (MITIP), 2008.

This paper focussed on the automation of the knitwear design process, as outlined in chapter 5. The paper also explained the relevance of CBR to knitwear design. A system was proposed whereby CBR was used to create designs, and if this failed then the CAD rules were used as a fall-back. It was suggested that adaptation would be knowledge-intensive, using transformation and substitution.

[97] Supporting Knitwear Design Using Case-Based Reasoning

Proceedings of the 19th CIRP Design Conference – Competitive Design, 2009.

The case representation was described, consisting of three levels of detail (questionnaire, sketch and chart). The similarity algorithm was explained, this included a simple equation for normalising similarity values.[1] The conditional applicability of some attributes in the similarity calculation was explained. The paper described the way in which the user is able to enter preferences for similarity, and how these are converted into weights, as per section 6.2.

A new version of the classic 4REs cycle [1] was suggested; this had particular applicability to case merging. The steps in the cycle were: repartition, retrieve, reuse, revise, recompose, repair and retain. The repartition step involved decomposition, with recompose being the reverse of this. The repair step was were inconsistencies between the parts were remedied.

[1] This was not used as it normalised the values relative to one particular case, rather than globally.

[14] Hierarchical Case Based Reasoning to Support Knitwear Design

CIRP Journal of Manufacturing Science and Technology, 2010.

This journal article describes SEACOP as a hybrid system which represents cases on five levels of detail (specification, sketch, linked regions, chart and pattern).[2]

The similarity algorithm was explained in detail. A worked example of how similarity was calculated and then normalised was presented.[1]

Retrieval was described as involving individual pieces, with the result being formed by case merging. Garments which contributed a piece to the result were described as being placed in a set called the *matched pool*. Subsequent retrievals prioritised garments in the *matched pool* over the rest of the case base. This was to ensure that the correct balance was struck between accuracy and compatibility. If each piece comes from a different garment then this may mean that the similarities of the individual pieces are high; however it may be hard to ensure compatibility as the different pieces must fit together into one garment. If all the pieces come from the same garment then incompatibility is less of issue but the similarities of the individual pieces may be low. Resizing pieces was cited as a one method of repairing incompatibility between pieces from different garments.

Adaptation was described as being knowledge intensive and many examples of adaptation rules were given. The article also discussed how patterns how constraints are managed, and how patterns are created and edited as per chapter 5.

Not all the ideas presented in these three papers were implemented, but some were discussed in section 9.3. For example, the ideas about case merging can be combined with RDR, and have the potential to produce a system with even more accurate adaptation, whilst still having only a light knowledge engineering burden.

[2] The specification is termed the questionnaire in this thesis. Linked regions were not implemented. Chart production was automated but is not a part of the CBR. Patterns were left for further work.

Appendix A

Knitting stitches and patterns

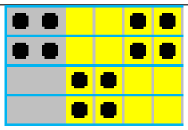
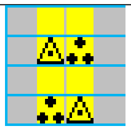
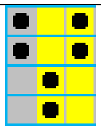
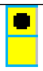
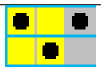
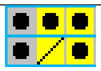
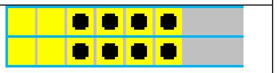
Name	*Knitting chart*
Box stitch	
Bramble stitch	
Double moss stitch	
Garter stitch	
Moss stitch	
Reverse stocking stitch	
Rice stitch	
Stocking stitch	
Rib (e.g. 2x4 rib)	
Other	as defined by the user

Table A.1: Standard stitch patterns

Abbreviation	*Size*	*Stitch count effect*	*Category*	*Instructions*	*Icon*
3from1	1	+3	bobble or knot	Knit 1, purl 1, knit 1) all in the same stitch	∴
b5	1		bobble or knot	Knot: (k1, p1, k1, p1, k1) in st to make 5 sts from 1, then pass 2nd, 3rd, 4th and 5th sts, one at a time, over 1st st	●
C3L	3		cable or twist	Slip next st to cable needle and hold to front of work; k2 from left needle, k1 from cable needle	
C3R	3		cable or twist	Slip next 2 sts to cable needle and hold to back of work; k1 from left needle, k2 from cable needle	
C4B	4		cable or twist	Slip next 2 sts to cable needle and hold to back of work; k2 from left needle, K2 from cable needle	>/ /<
C4F	4		cable or twist	Slip next 2 sts to cable needle and hold to front of work; k2 from left needle, k2 from cable needle	>\ \<
C5B	5		cable or twist	Slip next 3 sts to cable needle and hold to back of work; k2 from left needle, k3 from cable needle	
C5F	5		cable or twist	Slip next 3 sts to cable needle and hold to front of work; k2 from left needle, k3 from cable needle	
C6B	6		cable or twist	Slip next 3 sts to cable needle and hold to back of work; k3 from left needle, K3 from cable needle	
C6F	6		cable or twist	Slip next 3 sts to cable needle and hold to front of work; k3 from left needle, k3 from cable needle	
castoff	1		cast on/off	Cast off 1 st	▬
caston	1		cast on/off	Cast on 1 st	
hole	1	+1	increase	Yarn forward and over needle to make a st	○
incr1p	1	+1	increase	Increase 1 st by working p in back then front of st	▿
k_p	1		standard	K1 on right-side rows, p1 on wrong-side rows	
k1tbl_p1tbl	1		standard	K1 tbl (aka KB1) on right-side rows, p1 tbl (aka PB1) on wrong-side rows	╱
k1tblws	1		standard	K1 tbl on wrong-side rows	╳
k2tog_p2tog	1	-1	decrease	K2 tog on right-side rows, p2 tog on wrong-side rows	◿
k2togtbl	1	-1	decrease	K2 tog tbl	◺
k3tog	1	-2	decrease	K3 tog	▲
k4tog	1	-3	decrease	K4 tog	◢
k4togtbl	1	-3	decrease	K4 tog tbl	◣

Table A.2: List of stitches used

Abbreviation	*Size*	*Stitch count effect*	*Category*	*Instructions*	*Icon*
largebobble	1		bobble or knot	Large bobble: (k1, p1, k1, p1, k1) in st to make 5 sts from 1, turn, p5, turn, k5, turn, p5, turn; pass 2nd, 3td, 4th and 5th sts over 1st st then k in back of this st	⊡
largeknot	1		bobble or knot	Large knot: (K1, p1, k1, p1, k1, p1, k1) in st to make 7 sts from 1, pass 2nd, 3rd, 4th and 5th sts, one at a time, over 1st st then k in back of this st	◉
m1	1	+1	increase	Make a st by picking up strand in front of next st and p it in back	∪
m1p	1	+1	increase	Make a st by picking up strand in front of next st and p it in back	⊍
m2	1	+2	increase	Make 2 sts by picking up strand in front of next st and p it in back	
m2p	1	+2	increase	Make 2 sts by picking up strand in front of next st and p it in back	
p_k	1		standard	k1 on wrong-side rows	●
p2tog_k2tog	1	-1	decrease	P2 tog on right-side rows, k2 tog on wrong-side rows	◿
p2togtbl	1	-1	decrease	P2 tog tbl on right-side rows	◺
p3tog	1	-2	decrease	P3 tog	△
purlbobble	1		bobble or knot	Purl bobble: (p in front, back, front, back, front) of st to make 5 sts from 1, turn, k5, turn, p5, turn, k5, turn; pass 2nd, 3rd, 4th and 5th sts, one at a time, over 1st st then k in back of this st	○
s1k1psso_p2togtbl	1	-1	decrease	Skpo on right-side rows, p2 tog tbl on wrong-side rows	◺
s1k2togpsso	1	-2	decrease	Sl 1 knitwise, k2 tog, psso	⋀
s1p	1		standard	Sl 1 st purlwise, taking yarn behind work	❘
s2ask2togk1psso	1	-2	decrease	Sl 2 sts as if to work k2 tog; k1, psso	⋔
sl1	1		standard	Slip the next stitch onto the right hand needle without knitting it	¦
slleftonright	1		standard	St left on right-hand needle after casting-off	✳
smallbobble	1		bobble or knot	Small bobble: (k1, p1, k1, p1, k1) in st to make 5 sts from 1, turn, p5, turn; pass 2nd, 3td, 4th and 5th sts over 1st st then k in back of this st	◙
smallbobblek5	1		bobble or knot	Small bobble k5: (k1, p1, k1, p1, k1) in st to make 5 sts from 1, turn, k5, turn; pass 2nd, 3td, 4th and 5th sts over 1st st then k in back of this st	◙
T2B	2		cable or twist	Slip next st to cable needle and hold to back of work; k1 from left needle, p1 from cable needle	
T2F	2		cable or twist	Slip next st to cable needle and hold to front of work; k1 from left needle, p1 from cable needle	

Table A.3: List of stitches used, continued

Appendix B

Chart Tool

SEACOP realises the detail of the design as a knitting chart. The chart stage is equally as important as the sketch since it explicitly determines which stitches the knitter must use to produce the garment. Many knitters are able to take a knitting chart and produce a garment from this, without requiring any knowledge of knitwear design. In contrast, a sketch on its own would be of limited use to a non-designer. For a definition and introduction to the concepts involved in knitting charts, see section 4.5.

The way that charts are dealt with in SEACOP is explained below, starting with a description of the user interface. The user is provided with the means to view and edit charts. They cannot create new charts but this is done for them by SEACOP, by using simple rules that take the sketch and discretise it as knitting stitches. Finally, section B.3 explains how SEACOP ensures that charts are correct, using rules that tally with established conventions in knitting.

B.1 User Interface

Figure B.1 shows an example of the chart user interface. In this example, the chart (shown on the right) is for the front of a sweater with a bottom border. The bulk of the garment is stocking stitch, and the border is a rib stitch. SEACOP shows the patterns in different colours so that the user is able to distinguish between them. These colours do not necessarily correspond to the colours of the actual yarns used. However, as section B.1.1 explains, the user can change the colours. The user is able to edit the chart via right-click menus, and this is detailed in section B.1.2.

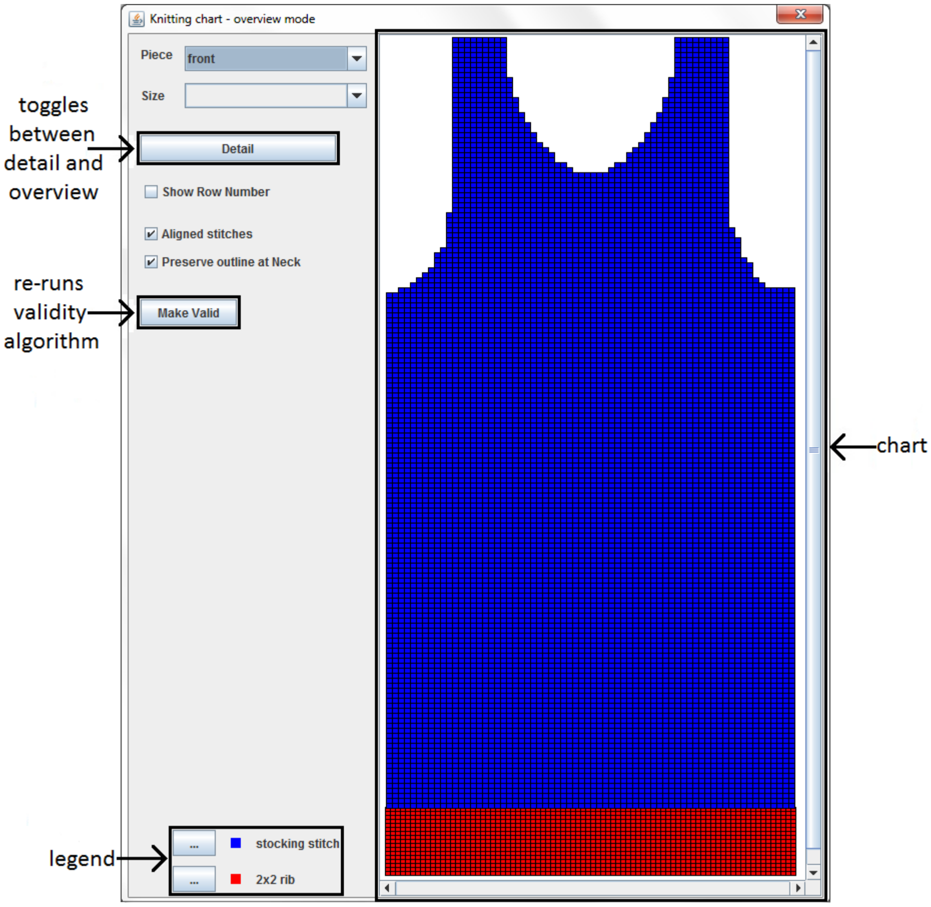
B.1.1 Changing preferences

The controls on the left hand side of the chart window allow the user to change a number of preferences. The topmost option allows them to change which piece is viewed. Below this, the size option allows the user to select the size shown; by default the smallest size is shown.

The Detail button in figure B.1 allows the user to toggle between *detail* and *overview* modes. Overview mode is designed to allow the whole piece (or a substantial part of a piece) to be visible in the window (as in in figure B.1), however individual stitches are simply shown as plain rectangles. In detail mode (shown in figure B.2), icons are shown to indicate the composition of individual stitches. Only stocking stitch is shown as plain rectangles, as per an established convention.

If the *show row number* box is ticked, then an integer is displayed to the right of each row, indicating the row number; this is useful for any issues with correctness (see below). The *aligned stitches* box controls the presentation of the stitches; if the box is ticked then stitches are aligned in columns wherever possible. If it is not ticked then stitches are aligned as closely to their position in the sketch as possible.

When *aligned stitches* is ticked, the *preserve outline at neck* option becomes available. This option controls the positioning in the row immediately above the neck divide, as shown in figure B.3. If the box is ticked (as it is by default) then the shaping at the edges will be consistent with that in the outline of the sketch. Otherwise, the rounding error between the chart and the sketch will be distributed equally


Knitting chart - overview mode

Piece: front

Size:

□ Show Row Number

☑ Aligned stitches

☑ Preserve outline at Neck

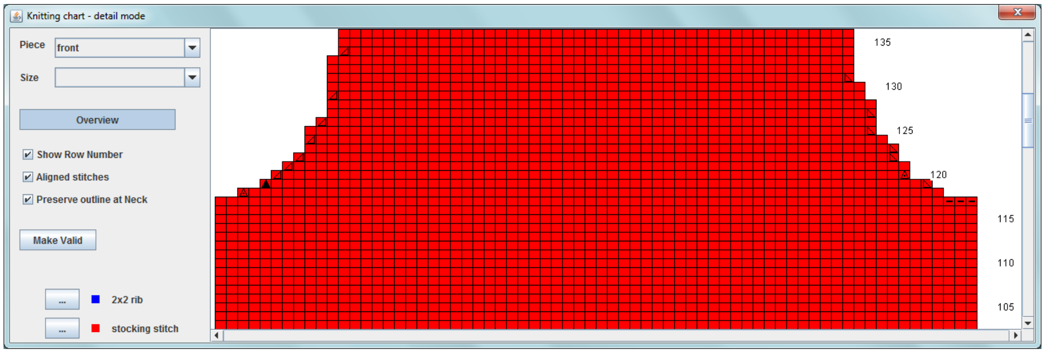
toggles between detail and overview

re-runs validity algorithm

legend

chart

Figure B.1: Chart tool showing a sweater with a bottom border


Knitting chart - detail mode

Piece: front

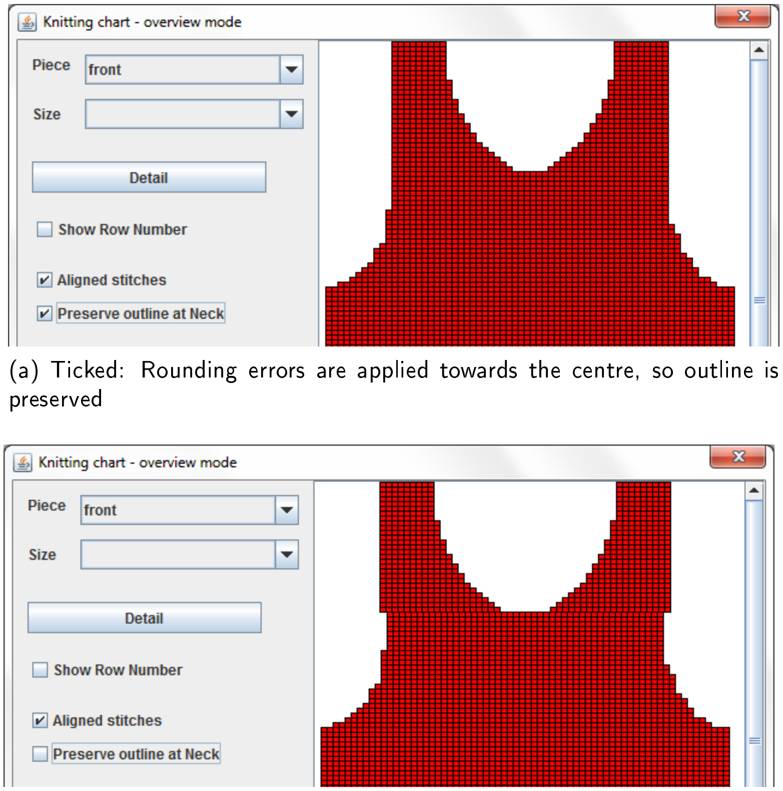
Size:

☑ Show Row Number

☑ Aligned stitches

☑ Preserve outline at Neck

Figure B.2: Detail mode showing the row number


Knitting chart - overview mode

Piece: front

Size:

Detail

□ Show Row Number

☑ Aligned stitches

☑ Preserve outline at Neck

(a) Ticked: Rounding errors are applied towards the centre, so outline is preserved

Knitting chart - overview mode

Piece: front

Size:

Detail

□ Show Row Number

☑ Aligned stitches

□ Preserve outline at Neck

(b) Not ticked: Rounding errors are applied evenly at either end

Figure B.3: Effect of preserve outline at neck

either side of each group of portions. The effect of having the box unticked is that the integrity of the outline will be broken, but the size of the gap for the neck will be more realistic.

Towards the bottom of the window is the legend. Although only the stitch patterns are displayed, if the user clicks the adjacent button the full information about yarn, tension, colour and needle is available. The user is offered the opportunity to change the colours.

Correctness

The chart is generated (as per section B.2) before the chart window is displayed. The correctness algorithms are subject to limitations (described in section B.3.5). If the chart cannot be made completely valid or there is a shaping problem, an advisory message is displayed with the row number so that the user can manually make the required corrections.

The user can re-run the validity algorithm by pressing the *Make Valid* button. This is particularly useful if they have edited a row and want to ensure the chart is still valid. If the algorithm cannot ensure validity then an advisory message is displayed.

B.1.2 Editing Charts

Ideally, the charts that are automatically generated should be what is required. However, the rules used to generate charts are not perfect and it is impossible to envisage every possible combination of garment. Similarly, the correctness algorithms have limitations. Therefore, SEACOP includes the capability for users to manually edit charts. The editing operations have a fairly localised scope, since the idea is that if major changes are required (e.g. adding 3cm to the length), then they would be accomplished in the sketch tool.

If the user right-clicks on a stitch it is highlighted and a pop-up menu appears: this has Insert, Delete, Edit and Reset options. There is then a further option to choose the scope of the operation: it can affect a single *Stitch* or a whole *Row*. The exception to this is Reset: this is only applicable to a whole Row.

	Stitch	*Row*
Insert	User must choose the new stitch, all stitches are available. Also they must pick whether the new stitch is to the right or left of the highlighted one.	The user must choose whether the new row is above or below the highlighted cell.
Delete	The user must choose whether the remaining stitches are shifted to the left or to the right.	No further choices are necessary.
Edit	User must choose the stitch; all stitches are available.	The user must choose the stitch; the row will then consist solely of the stitch the user has chosen. Shaping stitches and cable stitches are not available.
Reset	Not applicable.	No further choices are necessary.

Table B.1: Options for editing charts

Then, the user may be asked to make further choices, as per table B.1. An example is inserting a stitch, as per the example shown in figure B.4. Note that when editing a row, shaping and cable stitches are excluded because it would not make sense to have a whole row of these.

B.2 Generating Charts

The chart generation process starts with a single sketch and produces a knitting chart. It automates a particularly tedious stage in pattern production, in a consistent and repeatable manner. There can be several charts corresponding to the same sketch, and to different sizes. Due to the limitations of the sketch tool, and hence the chart generation process can make few assumptions about what sort of a sketch it will receive as its input. However, using a series of common sense heuristics, it is able to cope with most sketches.

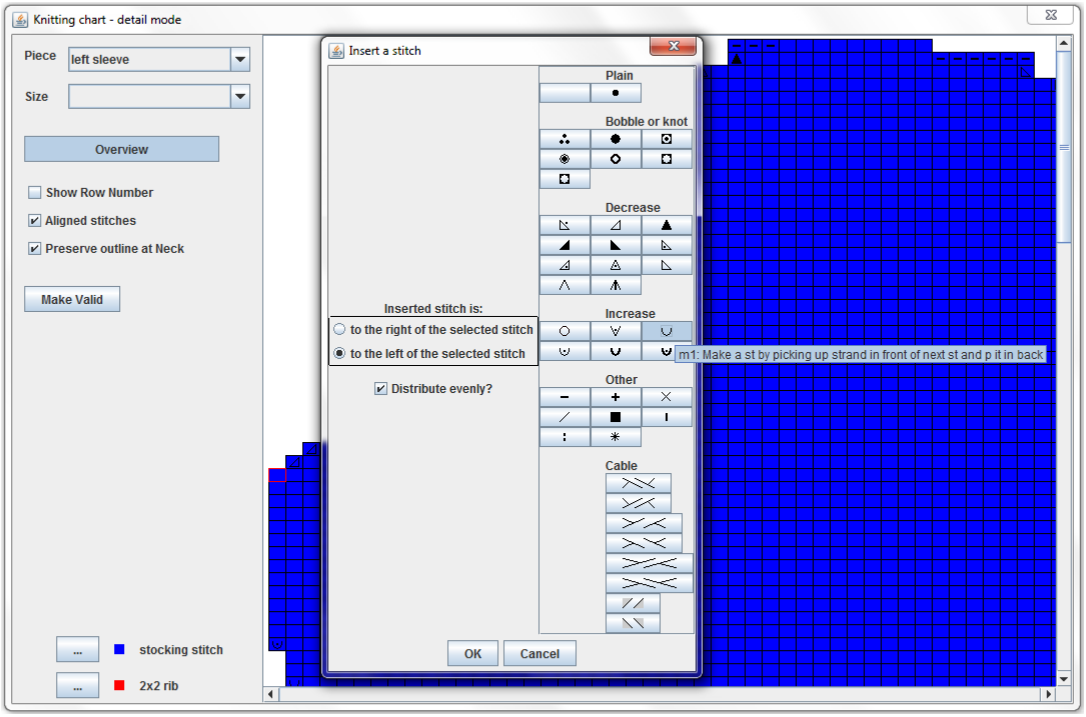
During chart generation, various intermediate objects are generated. Firstly, undirected graph representations of the sketch (known as the *simpler graph* and *full graph*) are produced. Cycles from these graphs are extracted so that each cycle (which is known as a *cycle-polygon*) corresponds to a polygon in the sketch. Each polygon is examined to determine which stitch pattern lies within it, in a process called *coding*. Then, any ambiguities about row tension are resolved through the creation of objects called *tension boxes*. Some areas are marked as being 'no cut areas', which means that the integrity of the pattern must be preserved, i.e. the width of these areas must be a discrete multiple of the pattern repeat width.

In the latter stages of chart generation, the composition of the stitches is ascertained. To begin with, the pattern is just naively repeated across the shape. Then, algorithms are invoked to make the chart correct, as per section B.3.

B.2.1 Simplified representation for identifying polygons

An important prelude to generating the sketch is the production of an alternate, simplified representation of the sketch. The simplified representation is in the form of an undirected graph which hides (but does not remove) unnecessary complications in the sketch, facilitating the polygon discovery algorithm that is described below.

As explained in section 5.2.5.1, the sketch contains a list of shape points. Each shape point includes both a purpose and a location and acts as an instruction on how to draw a part of the sketch on the screen.


Knitting chart - detail mode

Piece: left sleeve

Size:

☐ Show Row Number

☑ Aligned stitches

☑ Preserve outline at Neck

Insert a stitch

Inserted stitch is:

○ to the right of the selected stitch

● to the left of the selected stitch

☑ Distribute evenly?

m1: Make a st by picking up strand in front of next st and p it in back

🟦 stocking stitch

🟥 2x2 rib

Figure B.4: Inserting a stitch at the edge of a sleeve in detail mode

The list of shape points is loosely analogous to a directed graph, with each shape point representing both an arc (the purpose) and a vertex (the location). This is very sensible from the point of view of rendering sketches on the screen, but it is not a form which is suitable for discovering polygons, an activity which is necessary to determine the areas of the knitting that will be covered by the different stitch patterns.

Constructing the undirected graph

Firstly, for every location in which there is a shape point in the list (for a particular piece), a *planar point* is created. Whilst a shape point has both a location and purpose, a planar point consists of a location and a set of zero, one or two shape points. The set will consist of the shape points which are in the same location as the planar point; there will usually be one but if there are co-located points (see section 5.2.5.1) then there may be two. The purpose of the set is to provide a link between the sketch and the undirected graph representation; this link will be important during coding, as described in section B.2.3.

The location of the planar point will be the same as that of the equivalent shape point(s), except if the piece is a sleeve. The sleeves are laid out horizontally on the sketch, since this is the best use of space on the screen. However, charts are traditionally shown with the direction of knitting being horizontal, so for sleeves the location is subject to a 90º anticlockwise rotation.

A graph corresponding to the sketch is then constructed. The vertices in the graph are planar points, and the edges have one of three possible labels: *line*, *curve* or *redundant* (see figure B.5):

- A *line* edge corresponds directly to a line segment from the sketch.
- A *curve* edge links two planar points that are created when a curve is discretised. Curves are discretised because constructing the directed acyclic graph (DAG) in section B.2.2 involves checking if a polygon is inside another polygon. This is considerably easier to implement than testing if a shape involving curves is inside another similar shape.
- A *redundant* edge is equivalent to a series of curved edges, and is used in the polygon discovery (see below).

The list of shape points is processed and when the *n*th is encountered:

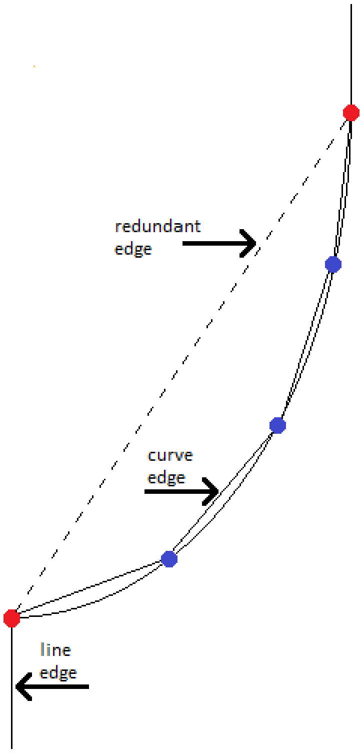


Figure B.5: Line, redundant and curve edges

1 If the *n*th shape point is the end of a line then a *line* edge is added to the graph between the planar points corresponding to shapepoints *n* and *n*-1.

2 If the *n*th shape point is for a vertical or horizontal band, then two new planar points are created[1], corresponding to the ends of the line that is making up the band. Each planar point will have the nth shape point in its set. A *line* edge is added to the graph between the two new planar points.

3 If the *n*th shape point is part of a Bézier curve, arc or neck band then the curve is discretised as a series of straight lines. New planar points are created[1] for the intersections of these lines, and they are connected with *curve* edges. The first and last planar points will have the corresponding shape point in their set, but the intermediate ones will have an empty set. Then, a *redundant* edge will be inserted between the first and last shape points. Figure B.5 shows a portion of a sketch which illustrates this; three[2] new planar points (coloured blue) have been added to facilitate the discretisation of the curve. The redundant edge is shown as a dashed line.

4 If the *n*th shape point is a button, then it is ignored, as buttons do not feature in the charts.

The next stage is where intersect points are dealt with. The fundamental constraint about non-intersects ensures that the end of a line segment will always intersect with another line segment (or curve, but curves have been discretised for these purposes). The graph will be modified for each planar point p_x which corresponds to a leaf vertex[3]. p_x will lie on another line segment that the line it terminates intersects with. This line segment will correspond to a line edge in the graph which is incident to vertices p_n and p_{n+1}. The edge from p_n to p_{n+1} will be removed and replaced with two line edges. One of these is incident to p_n and p_x, and the other is incident to p_x and p_{n+1}. Figure B.6 shows an example where the newly created edges are labelled *a* and *b*. Beforehand, p_n and p_{n+1} would have been linked with one edge, *ab* (not shown).

[1] For a sleeve, the planar points that are added in steps 2 and 3 will be subject to a 90º anticlockwise rotation.

[2] This is for illustrative purposes only; typically more than three line segments would be used to discretise a curve.

[3] A leaf vertex has a degree of 1.

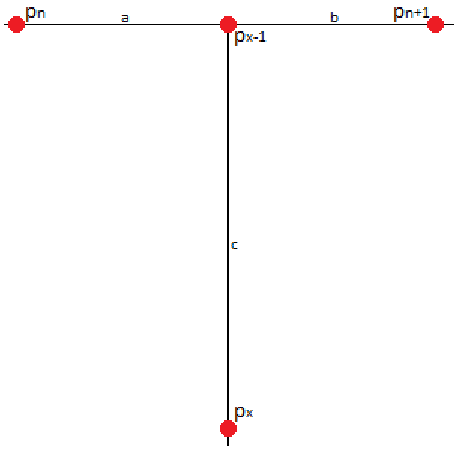


Figure B.6: Dealing with leaf vertices via intersects

Object		*Sketch*	*Full Graph*	*Simpler Graph*
Edge	line		8	8
	curve		66	
	redundant		4	4
	total edges		74	12
Shape Points		20		
Planar Points			74	12

Table B.2: Statistics for a simple sweater front

The end result of the process described above is an undirected planar graph, called the *full graph*. In the sketch, shape points can have a variety of purposes, they can be part of a line or a curve or neither; and if they are part of a curve they are not necessarily at an end point of that curve. However the *full graph* is more straightforward since it consists of just two types of entity: planar points and their connecting edges. The addition of edges to leaf vertices (as shown in figure B.6) ensures that each vertex has a degree of at least two, which is important for finding cycles in the polygon discovery.

Polygon discovery

A copy is made of the *full graph*, from which all of the curve edges are removed from the *simpler graph*, leaving only *line* and *redundant* edges. The copy will be known as the *simpler graph*. Then, the set of simple cycles[4] in the *simpler graph* are found. Since each redundant edge is equivalent to a series of *curve* edges, the simpler graph is equivalent to the full graph with the redundant edges removed, but it has far less vertices and therefore computations such as cycle detection are far more efficient. Table B.2 gives an indication of the sizes of the graphs for a simple sweater front (as per figure 5.25a).

Each cycle that has been found will be referred to as a *cycle-polygon*. Each cycle-polygon consists of a list of planar points. Any redundant edges that are present in the cycle-polygons are replaced with their equivalent series of *curve* edges, from the full graph. The set of cycle-polygons which results from this process will be known as the *shape set*. With the redundant edges removed, the cycle-polygons in the *shape set* correspond as close to possible (given that the curves are discretised) to shapes in the sketch. This is important as cycle-polygons will (later in the chart generation process) give rise to regions of knitting stitches.

[4] A simple cycle is a path which has the starting and ending vertices the same; no other vertices may be repeated and the edges are all unique.

B.2.2 Understanding the geometry of cycle-polygons

Constructing the DAG

The next stage in the chart production algorithm is the production of a directed acyclic graph (DAG) of cycle-polygons. The *shape set* is examined; each pair of cycle-polygons $\{C_A, C_B\}$ is checked to ascertain their geometric relationship. Three types of relationship are possible:

- *inside-connected*: C_A is inside of C_B and they have at least one planar point in common.
- *inside*: C_A is inside of C_B but they have no planar points in common
- *separate*: where neither *inside-connected* nor *inside* applies

If the relationship is not separate, then C_A and C_B are added to the graph as vertices; an arc is added between them and this is labelled either *inside* or *inside-connected* as appropriate. When all pairs have been examined, the DAG is simplified to remove redundant arcs: if there is an arc from C_A to C_B and C_B to C_C then the arc from C_A to C_C is redundant.

Establish which cycle-polygons need coding

Coding is the process of assigning a stitch pattern to a region of knitting. The next step is the identification of the *codable set*, which consists of cycle-polygons which need *coding*.

- All vertices with an outdegree[5] of zero are included in the codable set. These leaf vertices have nothing inside of them and so correspond directly to a region of knitting which will have only one stitch pattern.
- Where an edge labelled with *inside* is incident to a vertex V, then V is added to the codable set. V will have an indegree[6] of one, due to the removal of redundant arcs which was described above.

Cycle-polygons which are not added to the codable set will consist of "composite regions", which have an outdegree greater than one. Such cycle-polygons are not coded since they potentially consist of a mix of stitch patterns; their direct successors will be coded instead.

B.2.3 Coding

Assigning the codes

Each *code* is associated with a feature of the pattern that is associated with a stitch pattern. Most shape points in the sketch will carry a *code*. The *codes* are added when the shape points are created, during the sketch instantiation algorithm. The associated feature could be:

- one of the patterned pieces: the collar or hood
- a patterned element: a particular panel or pocket, or the region behind a pocket
- a region on the body: neck band, bottom border, front border or yoke
- the cuff
- a region which is knitted in the background stitch.

[5] The outdegree of a vertex is the number of arcs leading away from it

[6] The indegree of a vertex is the number of arcs leading towards it

Coding is the process in which a *code* is assigned to a cycle-polygon (known as C). The principal use of the codes is that they indicate which regions of the piece are to be knitted in which stitch patterns and at what tension. The first step in algorithm B.1 is the enumeration of the frequencies of the set of *codes* which are carried by the shape points that are linked to the planar points that make up C (this set is known as T). Coding first attempts to find a code from T by utilising the frequency of those *codes*. For example, C may be a pocket region delineated by four planar points; these planar points will be associated with shape points. If all of these carry a *code* for the pocket region then T will consist of just that one code and as the situation is ambiguous then the cycle-polygon is coded as a pocket region.

However, sometimes the situation may not be so clear-cut; for example in a cardigan front with a front border, the cycle-polygon that makes up the largest part of the front may carry codes for both the background stitch and front border. If T contains more than one code then algorithm B.1 resolves the situation, first by checking if C is part of larger region that has been previously been coded as per the previous paragraph (as it was unambiguous); in this case the code for the "container" region is used. Otherwise, the relative frequency of the codes in T is utilised; for example if 8 shape points are coded as background stitch and only 2 as the front border, then the former would be applied as it is more common within C.

Algorithm B.1 is designed to produce an output in all circumstances. In the very unlikely event that none of the points are coded (i.e. T is an empty set), the background stitch is applied.

If the situation is still unresolved then by definition there must be a tie: a subset of T (called S) will be equally prevalent. The tie is broken by choosing the code (from S) which occurs the least amongst cycle-polygons that have been previously coded. For example, if five shape points are coded for the yoke and five for the background stitch, and a different cycle-polygon has previously been coded as yoke but none have been coded as the background stitch, then the background stitch would apply.

Post-coding processing

The codes are used to facilitate some further processing which are prerequisites for the latter stages of the chart generation process. Firstly, the cycle-polygon corresponding to each pocket is copied. The new cycle-polygon is given a special tag to indicate it is 'behind the pocket'. As this region of knitting is not seen, it will be knitted in the background stitch, irrespective of the pattern used on the pocket. When the chart is viewed, the *behind pocket* regions will be the ones that are visible, since those correspond to the ones that are actually adjacent to their neighbours in the knitting.

Secondly, redundant cycle-polygons are removed from the DAG. A redundant cycle-polygon is one which has the same tag as its direct predecessor[7]. Cycle-polygons tagged as neck band or front border are not shown in the chart tool. The front border is usually knitted in a perpendicular direction to the rest of the garment, and is normally a thin strip of material with no special features, so its chart would be quite simple anyway. The neck band is not shown as the geometry of its corresponding cycle-polygon does not correspond to reality. Although the band may be curved, in reality it is more likely to be a trapezium[8] in shape, as fabric stretches.

The algorithms for shaping (as described in section B.3) assume that they are working with a continuous piece of knitting, and this will not be the case for the cardigan front. So, if necessary, the DAG is decomposed into a set of *weakly connected digraphs*. There will be one element in this set, unless the piece happens to be the front of a cardigan, in which case there will be two. The purpose of the decomposition is to be able to distinguish the two halves of a cardigan front.

B.2.4 Assigning row tensions

The previous sections have shown how the sketch can be mapped to a series of regions known as cycle-polygons. The latter stages of the chart generation process are concerned with how these cycle-polygons

[7] A vertex P is the direct predecessor of vertex Q if there is an arc from P to Q.

[8] The traditional British definition of a trapezium is used in this thesis: a convex quadrilateral with one pair of parallel sides.

Algorithm B.1 Coding

```
Procedure: coding
Inputs: codable set, DAG

for each cycle-polygon C in the codable set
  let T be the set of tags carried by the planar points in C
  enumerate the frequencies in which elements of T occur in C
  if T has a cardinality of one then
    apply this tag to C
  else
    search DAG to see if C is contained within a cycle-polygon
    that is already coded

    if search was successful then
      apply the tag of the container to C
    else
      if one tag in T has a higher proportion than others then
        apply the tag with the highest proportion to C
      else
        if T is an empty set then
          apply the background stitch tag to C
        else
          if one tag in T is the scarcest from already coded cycle-polygons
          then
            apply the scarcest tag to C
          else
            the choice of tag is arbitrary from within T
          end if
        end if
      end if
    end if
  end if
end for
```

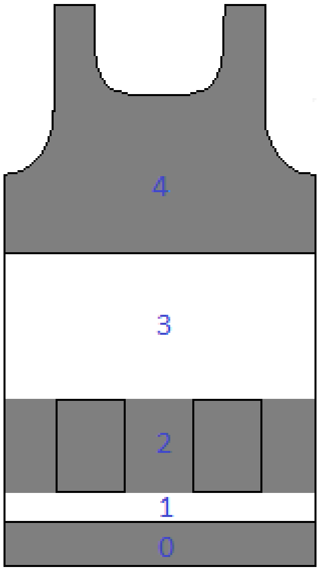


Figure B.7: Tension Boxes

can be used to determine the composition of stitches. The *cycle-polygons* can have different tensions and can potentially exist side-by-side; the problem with the latter configuration is that in SEACOP, we assume that *each row has a uniform row tension*[9]. If this assumption were to be violated, portions would be rendered on the screen with different heights, and as the effects of this would be cumulative, the user might not be able to distinguish which portions were part of which rows.

In cases where there is only one *cycle-polygon*, or all the *cycle-polygons* have the same row tension, then the situation is unambiguous. Otherwise, a heuristic is needed to derive an *assumed row tension*. For each piece (or half of the piece, in the case of a cardigan front), its set of *cycle-polygons* is examined. The y-coordinates of the vertical extremities of each *cycle-polygon* are listed and the list is sorted. For a list of coordinates y_0, y_1...y_n there are $n-1$ intervals; each interval is known as a *tension box*. Where $0 \leq a \leq n-1$, each *tension box* t_a is associated with a set of *cycle-polygon*s, all of which contain at least one planar point p with a y-coordinate in the range $[y_a, y_{a+1}]$. Figure B.7 shows an example where $n = 5$.

A *tension box* is associated with a series of *rows* of knitting. Since every row in the *tension box* is likely to consist of the same set of *cycle-polygons*, it is reasonable to apply the same *assumed row tension* to every row that is part of the *tension box*.

The important step in deriving the *assumed row tension* is the determination of the *most prevalent subset* (of *cycle-polygons*). The *most prevalent subset* is intended to consist of the *cycle-polygons* which have the largest area that is included within the region delimited by the *tension box*. The rationale is that these *cycle-polygons* should be the ones which actually determine the *assumed row tension*; if a large part of an area has one tension then this may actually apply since most yarns are capable of stretching to some extent.

The shapes associated with the *cycle-polygons* are not necessarily rectilinear and could be concave, so an exact determination of the areas is non-trivial. Therefore, a simple heuristic is employed. Three notional lines are defined as being on the 5th, 50th and 95th percentile of the interval associated with the tension box. The line segments which are formed when these notional lines cross the *cycle-polygons* are examined and the sums of the lengths of the line segments associated with each cycle-polygon are enumerated. The cycle-polygons which are associated with the greatest line length(s) are members of the *most prevalent subset*. Often there will be just one *cycle-polygon* in this set. However, in general the *assumed row tension* is deemed to be the greatest row tension of those *cycle-polygons* in the *most prevalent subset*. Thus, all stitches are assumed to be as tall as the tallest; given that most yarns are capable of some stretching this is not unreasonable.

[9] There is no such assumption made about the stitch tension: stitches in a row can vary in width.

B.2.5 Creating portions

A row consists of a series of *portion*s, each of which is associated with its own stitch pattern and tension. The stitch pattern and tension are obtained from the *cycle-polygons* as follows.

For each *tension box* (see section B.2.4), where x is the *assumed row tension* expressed as the number of rows to a centimetre, and y_a and y_{a+1} are the vertical bounds of the tension box, then the number of *rows* (s) in a tension box will be:

$$s = round\left(x\left(y_{a+1} - y_a\right)\right)$$

The *rows* are assumed to be evenly distributed, since we assume a constant row tension inside a particular *tension box*. Therefore the (h) height of each *row* will be:

$$h = \tfrac{y_{a+1}-y_a}{s} \simeq \tfrac{1}{x}$$

For each *tension box*, a series of horizontal lines are defined by the arithmetic sequence shown below. Each horizontal line will correspond to the centre of a *row*;

$$y = y_a + \tfrac{h}{2},\, y_a + \tfrac{3h}{2},\, y_a + \tfrac{5h}{2} \,.....\, y_{a+1} - \tfrac{h}{2}.$$

Every time the line which corresponds to the *row* passes through a *cycle-polygon*, a *portion* is created. A *portion* is associated with a the stitch pattern (and tension) that is determined by coding the *cycle-polygon*. The line segment that lies inside that *cycle-polygon*[10] will be associated with that polygon and (later in this process) used to determine the number of stitches that the portion will hold, and the nature of any edge shaping. This line segment will be known as the *raw line*.

The next stage in the chart production process is where the number of stitches in a *portion* is determined; this is referred to as the *size* of the portion. Once this is known, then the stitches can be created and the knitting chart is then complete. The *size* is obtained by multiplying the length of the *raw line* by the stitch tension and rounding to the nearest integer.

Patterns which have edge stitches pose a complication. These patterns are designed to be applied with an integer number of repeats; further information about this is given in section 4.5. In order to preserve the integrity of these patterns, SEACOP creates a *no cut area* (NCA); within the NCA, the *size* is adjusted so that the number of stitches is a multiple of the number of repeat stitches (plus the number of edge stitches).

It is important that an accurate record of the position of each portion is kept, since this is needed for shaping, later in the chart production process. So, final adjustments are made to the *raw line*. The *adjusted line* is derived from the *raw line*. The first adjustment is to compensate for rounding errors. The *adjusted line* is stretched (from the dimensions of the raw line) until its length is equal to the combined width of the stitches in its portion. Finally, if necessary, surrounding *adjusted lines* are translated horizontally, to prevent the lines overlapping. When the line segments associated with two portions share an end-point, they are said to be *butted*. The main constraint on the stretch and translation operations are that if two portions are butted according to their *raw lines*, then they shall remain so according to their *adjusted lines*. To violate this might disrupt the composition of the *groups* (see section B.3.1).

[10] Since the cycle-polygons can be concave, care is taken in the implementation of this step in the process. Where the line intersects with the boundary of a cycle-polygon P, then a test is performed to ensure that the line segment that this delimits is actually contained within P. If it is inside, then a further check is performed to see if the line segment is contained within another cycle-polygon that is contained within P; this is repeated recursively until the smallest cycle-polygon that contains the line segment is found. If the line segment is not contained within P then a search is performed on the other cycle-polygons until the correct one (if any) is found.

Algorithm B.2 Determining the pattern row number

```
function: pattern-row-number
input:  garment row number g, stitch pattern S, number of rows
      in stitch pattern n

if g>0 and there is an equivalent portion on row g-1 then
   r:= the pattern-row-number of the equivalent portion + 1
else
   if g mod 2 = 0 then
      if the first row of S is knitted right to left then
         r := 0
      else
         r := 1
      end if
   else
      if the first row of S is knitted right to left then
         r := 1
      else
         r := 0
      end if
   end if
end if

output r mod n
```

B.2.6 Assigning stitches

The objective of the next phase in the chart production process is to assign stitches to the portions; thus a knitting chart is created, although it might not be a *correct* chart (but this is dealt with afterwards, as explained in section B.3). Fitting a pattern to a shape is one aspect of knitwear *grading*; the other is how the sketches are resized (as explained in section 5.2.7.2).

The computations in the previous sections have provided the *size* (number of stitches) of each portion, and the stitch pattern which applies to it. Stitch patterns have their own knitting chart composed of rows and stitches, as does the garment. The first step in setting the stitches for a particular portion *(P)* is: given the *garment row number (g)*, calculate the *pattern row number (p)*. SEACOP follows a convention in numbering the bottommost row of a knitting chart as zero, and therefore the topmost row in the stitch pattern will be numbered *n*-1, where *n* is the number of rows in that stitch pattern.

For rows in the garment other than the bottommost, algorithm B.2 begins by looking for an *equivalent portion (Q)* in row *g-1*. Two conditions must hold for portions to be equivalent:

- They must have the same stitch pattern and tension.
- The x-coordinates of the ends of the *raw line* of the two portions must be the same, or similar.

If *Q* is found then the *pattern row number* is that of *Q*, incremented by 1, modulo *n*. However there will be no equivalent portion if $g = 0$ or if, for example, *g* refers to the bottommost row of a rectangular panel; in these cases *p* is solely determined by *g*. There is an assumption that the bottommost row of the garment is always knitted right to left; therefore all rows where *g* is an even number will be knitted right to left and those where *g* is an odd number will be knitted left to right. However, no such assumption can be made about a knitting chart, and for the integrity of the pattern to be preserved the direction of knitting of the portions in *g* must be the same as that in row *p* of the relevant stitch pattern. If the direction of knitting of the bottommost row is the same as that in *g* then *p*=0; otherwise *p*=1 as the next row in the stitch pattern will be used.

If the portion P is part of a *no cut area* (see section B.2.5), then the edge stitches in *P* will be those from the *p*th row of the stitch pattern; the remaining stitches in the middle of the portion will contain

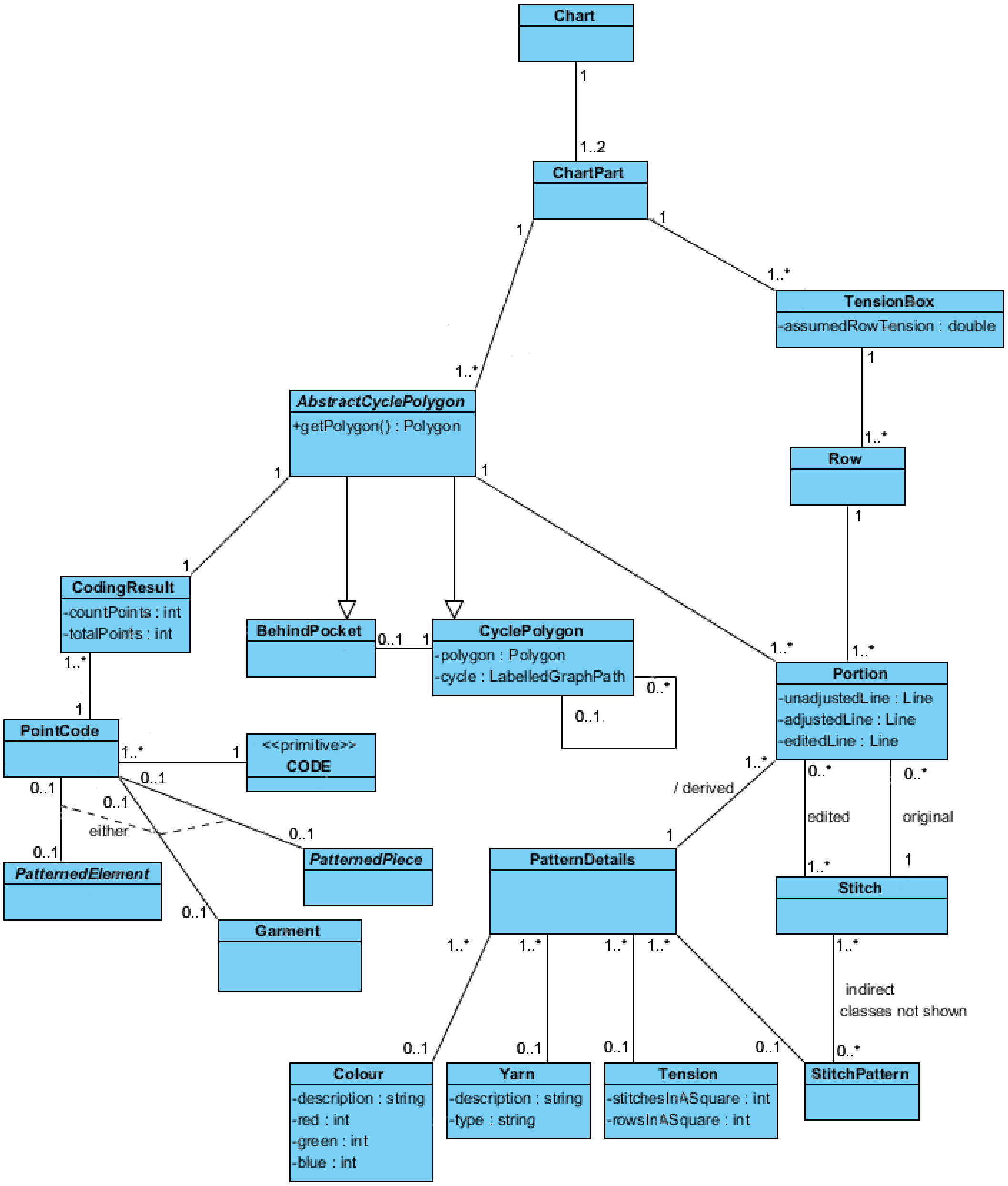


Figure B.8: Representation of charts

as many repeats of the pattern as needed. If the portion P is not part of a *no cut area*, the maximum number of integer repeats of the pattern are used for *P*. If there are any unassigned stitches at the edge, these are filled in with plain stitch (stocking stitch or reverse stocking stitch; see section 4.5).

B.2.7 Representation

After all the processes in section B.2 have been applied, a series of complex objects will have been created which enable a chart to be produced, with the correct tensions and stitch pattern. Figure B.8 shows the model that is created. There are some simplifications in the diagram, e.g. the representation of *no cut areas* and the graph is not included, and only a small number of the attributes are shown.

B.3 Correctness of Charts

There are two distinct (but related) aspects to the *correctness* of a knitting chart: *shaping*, and *validity*.

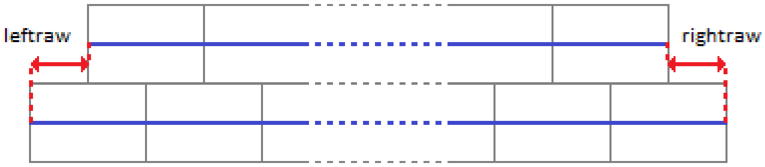


Figure B.9: Raw shaping

- *Shaping* uses shaping stitches (increases and/or decreases) to make the piece non-rectangular, for example to make an armhole or tapered sleeve. If the *shaping* in the chart is incorrect, then this will mean that the shapes in the finished garment are not in accordance with those in the sketch. In this case, the designer's intentions will not be fulfilled and the resulting garment may not be aesthetically pleasing and could even be unwearable (e.g. if the pieces cannot be sewn together).
- A *valid* chart is one in which any change in stitch numbers is balanced out by shaping stitches. An invalid chart will be syntactically correct (because it contains rows and stitches) but it will be semantically incorrect: *it does not correspond to a garment that could actually be knitted*.

Section B.2 explains how a knitting chart is produced from the sketch. The chart production process culminates in the determination of the row tension, stitch pattern, position (via the *adjusted line*) and the composition (i.e. the stitches) of each portion; this is what is required to produce a knitting chart. If the stitch patterns used in the garment contained no shaping stitches, and the piece was perfectly rectangular, then this would constitute a *correct* chart. However, this is not realistic since pieces of sweaters and cardigans are not rectangular, and shaping stitches (see section 4.4) are often used in stitch patterns to create lace or other effects.

Since the output of section B.2 is unlikely to be *correct*, SEACOP contains algorithms to detect and repair problems with both *shaping* and *validity*. The fundamental assumption in SEACOP's shaping algorithm is that *shaping stitches at the very edge of a group will produce the change in width necessary to make the chart consistent with the shapes in the sketch*. A description of how the shaping is calculated is presented in B.3.1. Applying the shaping is not completely straightforward due to the zigzag nature of knitting, but this is explained in section B.3.2.

The shaping and validity algorithm operates on a *group* of portions. A *group* consists of a series of portions, all of which are *butted* (see section B.2.5). Thus, each row will ordinarily consist of one or two *group*s; there will be two *groups* per row in the region at the top of the body, above the neck divide. Ordinarily, a group will have an equivalent *preceding* group in the row below it.

B.3.1 Determining the shaping

A group is normally shaped with reference to the *preceding* group, as per algorithm B.3. The terms used in this algorithm are explained in table B.3.

If there is no preceding group, then the shaping is deemed to be zero; this will be the case for the bottommost row. It will also apply where the neck divides[11].

Algorithm B.3 compares the line segments associated with the group (G) and its predecessor (P) to give the *raw shaping*, as depicted in figure B.9. The *raw shaping* is then *corrected* by including an *error* term which represents the difference between the ideal and actual shaping which was achieved in group P. If this rounding error is not carried forward, then the shaping in the chart tends to differ markedly from that in the sketch. This is particularly so in the tapered region of the sleeve, where the shaping often happens to be less than half a stitch, which would be rounded to zero unless the *error* is carried forward.

[11] The procedure for creating the neck involves casting off many stitches. This is outside of the scope of the shaping algorithm; the chart will not show these cast off stitches. Similarly, the user needs to create a cast on row at the bottom of the piece and to cast off all the stitches at the top. These situations are also outside of the scope of the shaping algorithm. However, since these are fairly standard processes, it would not be difficult to implement these things at the stage where the written pattern is produced from the chart. Production of written patterns is also outside of the current scope of SEACOP.

Algorithm B.3 Determining the shaping

```
inputs: a group G

if there is no preceding group to G then
   leftshaping(G) := 0
   rightshaping(G) := 0

   lefterror(G) := 0
   righterror(G) := 0
else
   P := the preceding group to G

   leftraw(G) := leftx(P) - leftx(G)
   rightraw(G) := rightx(G) - rightx(P)

   leftcorrected(G) := leftraw(G) - lefterror(P)
   rightcorrected(G) := rightraw(G) - righterror(P)

   leftexact(G) := leftcorrected(G) x lefttension(G)
   rightexact(G) := rightcorrected(G) x righttension(G)

   if all portions in G and P have the same stitch tension then
      stitchdiff(G) := numberofstitches(G) - numberofstitches(P)
   else
      leftdiff(G) := leftraw(G) x lefttension(G)
      rightdiff(G) := rightraw(G) x righttension(G)
      stitchdiff(G) := round(leftdiff(G) + rightdiff(G))
   end if

   exactsum(G) := leftexact(G) + rightexact(G)

   if exactsum(G)=0 then
      leftapportion(G) := stitchdiff(G)/2
      rightapportion(G) := stitchdiff(G)/2
   else
      leftapportion(G) := (leftexact(G) x stitchdiff(G))/exactsum(G)
      rightapportion(G) := (rightexact(G) x stitchdiff(G))/exactsum(G)
   end if

   leftshaping(G) := round(leftapportion(G))
   rightshaping(G) := round(rightapportion(G))

   lefterror(G) := leftshaping(G)/lefttension(G) - leftexact(G)
   righterror(G) := rightshaping(G)/righttension(G) - rightexact(G)
end if
```

Term	*Description*	*Unit*	*Integer?*
leftx	Leftmost x-coordinate on a line segment	cm	
rightx	Rightmost x-coordinate on a line segment	cm	
leftraw	*Raw shaping* on the left edge	cm	
rightraw	*Raw shaping* on the right edge	cm	
lefterror	*Error* on the left edge	cm	
righterror	*Error* on the right edge	cm	
leftcorrected	Shaping on the left edge, *corrected* for previous errors	cm	
righttcorrected	Shaping on the right edge, *corrected* for previous errors	cm	
lefttension	Stitch tension of the leftmost portion	stitches/cm	
righttension	Stitch tension of the rightmost portion	stitches/cm	
leftexact	Unrounded *exact shaping* on the left	stitches	
rightexact	Unrounded *exact shaping* on the right	stitches	
exactsum	Sum of the *exact shapings*	stitches	
numberofstitches	Count of the stitches	stitches	yes
stitchdiff	*Stitch difference*	stitches	yes
leftapportion	The part of stitchdiff allotted to the left edge	stitches	
rightapportion	The part of stitchdiff allotted to the right edge	stitches	
leftshaping	Finalised shaping on the left edge	stitches	yes
rightshaping	Finalised shaping on the right edge	stitches	yes

Table B.3: Terms used in algorithm B.3

Next, the *corrected* shaping is multiplied by the tension to give the *exact shaping*s, expressed in stitches. It is possible but not inevitable that the tensions of all the portions are equal. SEACOP makes the assumption that only the edge tensions are relevant, which is compatible with the main assumption stated in the opening paragraph of section B.3.

The next step in the algorithm is the calculation of the *stitch difference*. If the tensions in all portions of G and P are equal, then this is the difference in the number of stitches[12]. If tensions are unequal then the difference in the number of stitches may not actually correlate to the shaping at the edges; for example there may be a rectangular panel in the centre which has wider stitches and therefore there will be less stitches in G (compared to P), irrespective of the shaping. So, the *stitch difference* in this case is made by multiplying the *raw shaping* by the tension (of the relevant portion at the edge).

The goal of algorithm B.3 is that the shapings at the edge should add up to give the *stitch difference*. If this goal were not achieved, it could cause extra work for the validity algorithm. If the exact shapings cancel each other out, then a pragmatic approach is taken and the *stitch difference* is simply divided between both edges. Otherwise, the stitch difference is split between the edges in direct proportion to the value of the *exact shaping*s. The result of either calculation is rounded to the nearest integer, and then a check is performed to ensure that the goal has been achieved. Rounding errors are the only source of failure, and if necessary one of the values is rounded up or down by 1 in order to satisfy the goal. The choice of which value to change is determined by which would result in the lowest rounding error.

Finally, the *error* at each side is noted, ready for the next invocation of the algorithm. Then, the shapings are ready to be applied to the garment.

[12] Cable stitches are treated as being more than one stitch for these purposes as they are wider than standard stitches

		Shaping applies to	
		Start of group in row i	*End of group in row i*
Shaping strength	$n > 3$	add n cast on stitches to the end of row i-1	replace the last n stitches of row i with cast on
	$n \leq -3$	add n cast off stitches to the beginning of row i	replace the first n stitches of row i-1 with cast off

Table B.4: Shaping using cast off or cast on

B.3.2 Applying shaping

A general description of how shaping can be effected in knitting is given in section 4.4. The comments here explain how shaping is done in SEACOP.

For each piece in the garment, SEACOP starts with the bottommost row, working upwards, applying the shaping for a particular group once it has been determined (as per section B.3.1). The shaping is implemented by changing the edge stitches in a group (if necessary). A positive shaping will usually mean an increase stitch is placed at the edge, a negative shaping a decrease, and a zero shaping will mean that the edge stitch is not a shaping stitch. An appropriate stitch is chosen; for example, SEACOP might utilise k2tog ("knit two together") to make a decrease of 1, and k3tog ("knit three together") to make a decrease of 2.

In theory, infinitely powerful shaping stitches are possible. However, in practice stitches such as k4tog ("knit four together") are possible, but they are difficult for a knitter to accomplish. Therefore, it is common practice to implements shapings with a magnitude of 3 stitches or greater by casting off (as a means of decreasing) or casting on (as a means of increasing). The complication with this method is that it is not possible to cast off at the end of a row, and not standard practice to cast on at the start of a row. Therefore the actions are applied to the previous row instead, as explained in table B.4. Note that due to the zigzag nature of knitting, a particular side of the chart will alternate from being the start of the row to the end of the row, as discussed in section 4.5.

If SEACOP applies a shaping to the previous row, care is taken to ensure this does not nullify the effect of shaping which was previously applied to that row. This is relevant, for example, if the first stitch of row *i* is changed to single decrease, but then processing row *i+1* determines that the first 3 stitches of row *i* are to be replaced with a cast off. In this example, the first 4 stitches of row *i* would be replaced with cast-offs; the additional cast off stitch compensates for the 'lost' decrease.

B.3.3 Determining validity

A general description of validity and shaping stitches in knitting is given in section 4.5. The comments here explain how validity is determined in SEACOP.

After the shaping in a *group* has been determined and applied, the *group* is assessed for validity. The validity of *group* G is determined with reference to its *predecessor* (P). *Group* G is valid if it has a *discrepancy* of zero. The discrepancy is calculated thus:

$$discrepancy_G = stitches_G - stitches_P + castoffs_P - castons_G - shape_G$$

The terms are defined as follows:

- *stitches* - this is the count of stitches; cable stitches are wider than standard stitches and therefore count as more than one stitch

- *castoffs* - this is the count of the cast off stitches; if these are present they will have been added by the shaping algorithm

- *castons* - this is the count of the cast on stitches; if these are present they will have been added by the shaping algorithm

- *shaping* - this is the cumulative effect of shaping stitches, where increases are positive and decreases are negative.

If $discrepancy_G \neq 0$, then SEACOP invokes an algorithm to make the group valid, as described below.

B.3.4 Making a group valid

If the *discrepancy* of a group is non-zero, then SEACOP seeks to remedy this by modifying the *internal* stitches of a group. The *internal* stitches are those which are not involved in the shaping process, i.e. all stitches except the two edge stitches and any cast off or cast on stitches (which only occur near the edges).

The algorithm is an iterative one; the first step is to find the *optimum* internal stitch; this is the one that is judged as most suited to being substituted (for another stitch). This stitch is replaced by another stitch, which reduces the absolute value of the *discrepancy*. The replacement stitch will have a different "changing effect". The algorithm is repeated if necessary until the *discrepancy* is zero. The process is different depending on whether the discrepancy is positive or negative, as explained below.

Negative discrepancy

If $discrepancy_G < 0$, this means there are too few stitches in G. Algorithm B.4 is followed (without loss of generality, see below) to find the optimum stitch. The first priority is to locate a decrease stitch, since this stitch could be replaced by a non-changing stitch or reduced in strength. The next priority is to find a single increase; this could be turned into a double increase in order to increase the *discrepancy* by 1.

As a last resort, a non-changing stitch is sought since this can be turned into an increase. The reason why non-shaping stitches are given the lowest precedence is that shaping stitches tend to have a distinctive appearance and are often part of a pattern. Therefore, it is perhaps undesirable to introduce new shaping stitches unless there is no alternative.

If there is more than one decrease then they are prioritised according to algorithm B.5. The first priority is to find a decrease with a strength equal to the discrepancy; this stitch can be made to a non-shaping stitch and this will result in a zero discrepancy. If there is no such decrease then ones with a strength lower than the discrepancy; making a single decrease into a non shaping stitch will increase the discrepancy by one.

When other means fail, both algorithms B.4 and B.5 use the stitch that is closest to the end of the row to determine the optimum stitch. This stitch will still be an *internal* stitch, as the endmost ones are excluded from the inputs to these algorithms.

Positive discrepancy

In the event of a positive discrepancy, the inverse of algorithms B.4 and B.5 applies, by substituting "increase" for "decrease" and vice versa.

B.3.5 Limitations of correctness algorithms

SEACOP is capable of making most charts correct. However, the algorithms described above have limitations. Indeed it cannot be assumed that it is possible to devise perfect algorithms for the correctness of charts; such a problem may be undecidable (as in Turing's famous halting problem). Also, more elaborate algorithms (than are presented here) may cause the system to deviate from the user's intentions.

As explained previously, the correctness algorithms work on a piece by processing each row one at the time, starting at the bottom and working upwards; first the shaping is done and then the validity. If either algorithm encounters a problem, then the details of the problem are recorded, and the algorithm then attempts to continue with subsequent rows.

A problem is deemed to have occurred with the shaping algorithm if either leftraw and leftshaping or rightraw and rightshaping (as defined in table B.3) have opposite signs. This means one or both of the following must be true:

Algorithm B.4 Rectifying discrepancy<0

```
let D be the set of decreases
let cD be the cardinality of S
when cD is 1 then
        output the element in S
        terminate
when cD is 0 then
        let I be the set of single increases
        let cI be the cardinality of S
        when cI is 1 then
                output the stitch in I
        when cI is 0 then
                let N be the set of non-shaping stitches
                let cN be the cardinality of N
                if cN is 1 then
                        output the stitch in N
                else
                        output the stitch in N which is closest to
                        the end of the row
        otherwise (i.e. cI>1)
                output the stitch in I which is closest to
                the end of the row
otherwise (i.e. cD>1)
        output the priority decrease from D
```

Algorithm B.5 Prioritising decreases

```
input: a non-empty set of decreases (D)

let E be the subset of stitches with strength equal to discrepancy
let cE be the cardinality of E
when cE is 1 then
   output the stitch in E
when cE is 0 then
   let L be the subset of stitches with strength lower than discrepancy
   let cL be the cardinality of L
   when cL is 1 then
      output the stitch in L
   when cL is 0 then
      E must consist of stitches with a strength greater than discrepancy
      output the stitch in E which is closest to the end of the row
   otherwise (i.e. cL>1)
      output the stitch in L which is closest to the end of the row
otherwise (i.e. cE>1)
   output the stitch in E which is closest to the end of the row
```

- $(leftraw < 0 \land leftshaping > 0) \lor (leftraw > 0 \land leftshaping < 0)$
- $(rightraw < 0 \land rightshaping > 0) \lor (rightraw > 0 \land rightshaping < 0)$

This safeguard is intuitive since it does not make sense for the shaping that is applied to be the opposite of that which is derived from the sketch.

A problem is deemed to occur with the validity algorithm if none of the stitches in the row is capable of being transformed into a stitch that will reduce the magnitude of the discrepancy. In algorithm B.4, if $discrepancy_G < 0$, a problem would arise when D, I, and N were all empty sets. The possible causes of this are:

- If the group consists of only one or two stitches. Since the outer stitches are not changed, there is nothing the algorithm can do.
- If the internal stitches of the group consist of only double increases, and $discrepancy_G < 0$, or only double decreases where $discrepancy_G > 0$.
- If the group consists of only cast off stitches.

The solution to shaping and validity problems is the same: the user will have to manually edit the chart. For example, if the group consisted of just two stitches because it was at the very end of the piece, then those two stitches would have to be altered to make the chart valid, since validity takes precedence over shaping issues. Alternatively, it may be possible to fix the situation by changing the previous row.

Thus, even if the inbuilt algorithms cannot produce a valid knitting chart then manual methods exist whereby a competent user can complete the process. Since the user is able to re-run the validity algorithm at any time, is provides instant feedback as to the success of their editing.

Appendix C

Questionnaire Similarity Settings

Description	Code	*Rank*	*Weight*
The type of garment	SIM_GARMENTTYPE	priority	1
Has sleeves or is sleeveless?	SIM_HASSLEEVES	priority	1
Armhole style	SIM_ARMHOLESTYLE	priority	1
Shape of neck	SIM_NECKSHAPE	priority	1
Type of background stitch	SIM_STITCHBG	0	0.96
Fastener type	SIM_FASTTYLE	1	0.92
Waist fitting option	SIM_FITWAIST	2	0.88
Choice of neck option	SIM_NECKOPTION	3	0.84
Whether there is a front border or not	SIM_FRBORDER	4	0.8
Whether or not there is a yoke	SIM_YOKE	5	0.76
Whether there is a bottom border or not	SIM_BOTBORDER	6	0.72
The option for cuffs	SIM_CUFFS	7	0.68
Who is the intended wearer ?	SIM_WEARER	8	0.64
How many pockets there are	SIM_POCKETNUMBER	9	0.6
Position of the pockets	SIM_POCKETPOS	10	0.56
Straight part at the top of the sleeve	SIM_STRTOP	11	0.52
Straight part at the bottom of the sleeve	SIM_STRBOTTOM	12	0.48
Sleeve length	SIM_SLEEVELENGTH	13	0.44
Style of collar	SIM_COLLARSTYLE	14	0.4
Style of hood	SIM_HOODSTYLE	15	0.36
The stitch pattern used on the front border	SIM_FRBORDERSTITCH	16	0.32
The stitch pattern used on the bottom border	SIM_BOTBORDERSTITCH	17	0.28
The stitch pattern used on the collar	SIM_COLLARSTITCH	18	0.24
The stitch pattern used on the hood	SIM_HOODSTITCH	19	0.2
The stitch pattern used on the neck edging	SIM_NECKEDGESTITCH	20	0.16
The stitch pattern used on the yoke	SIM_YOKESTITCH	21	0.12
The stitch pattern used on the cuffs	SIM_CUFFSTITCH	22	0.08
Whether it is fully fashioned or not	SIM_FULLFASH	23	0.04
Width allowance	SIM_WIDTHALLOW	irrelevant	0
Length adjustment	SIM_LENGTHADJUST	irrelevant	0
Whether there are no superfluous pockets	SIM_POCKETSNOEXTRA	irrelevant	0
Number of buttons	SIM_BUTTONNUMBER	irrelevant	0
Button position	SIM_BUTTONPOSITION	irrelevant	0
Whether or not at least one of the sizes matches	SIM_ASIZEMATCH	irrelevant	0

Table C.1: Weights used for features

Table C.1 shows the descriptions and weights for the features. The remainder of the tables in this appendix give the "raw score" for the Likert settings; the key for this is table 6.3.

	Dropped	*Raglan*	*Semi set in*	*Set in*
Dropped	5	1	2	2
Raglan	1	5	2	2
Semi set in	2	2	5	4
Set in	2	2	4	5

Table C.2: Armhole styles

	V	*Round*	*Straight*	*Scoop*	*Slash*
V	5	3	1	3	2
Round	3	5	1	4	2
Straight	1	1	5	1	3
Scoop	3	4	1	5	1
Slash	2	2	3	1	5

Table C.3: Neck shapes

	Entire Front	*Top part only*	*Bottom part only*	*Middle only*
Entire Front	5	2	2	3
Top part only	2	5	1	2
Bottom part only	2	1	5	2
Middle only	3	2	2	5

Table C.4: Button positions

	Very Short	*Above Elbow*	*3/4 length*	*Wrist*
Very Short	5	3	2	1
Above Elbow	3	5	3	3
3/4 length	2	3	5	4
Wrist	1	3	4	5

Table C.5: Sleeve lengths

	stocking stitch	*reverse stocking stitch*	*moss stitch*	*garter stitch*	*bramble stitch*	*double moss stitch*	*box stitch*	*rice stitch*	*rib*	*other...*
stocking stitch	5	4	2	3	2	2	2	1	2	3
reverse stocking stitch	4	5	2	3	1	2	2	3	2	2
moss stitch	2	2	5	2	1	3	3	1	3	2
garter stitch	3	3	2	5	1	2	2	2	2	2
bramble stitch	2	1	1	1	5	1	1	1	1	1
double moss stitch	2	2	3	2	1	5	4	1	4	2
box stitch	2	2	3	2	1	4	5	1	3	2
rice stitch	1	3	1	2	1	1	1	5	1	1
rib	2	2	3	2	1	4	3	1	5	2
other...	3	2	2	2	1	2	2	1	2	5

Table C.6: Stitch patterns

	Baby	*Child*	*Man*	*Woman*
Baby	5	1	4	1
Child	1	5	1	1
Man	4	1	5	2
Woman	1	1	2	5

Table C.7: Wearers

	normal	*fitted*	*baggy*
normal	5	3	1
fitted	3	5	3
baggy	1	3	5

Table C.8: Waist options

	Buttons	*Zip*	*Belt*	*Other*	*None*
Buttons	5	2	2	1	2
Zip	2	5	2	1	3
Belt	2	2	5	1	2
Other	1	1	1	5	1
None	2	3	2	1	5

Table C.9: Fastener options

	none	*normal*	*loose*	*tight*
none	5	1	1	1
normal	1	5	3	3
loose	1	3	5	2
tight	1	3	2	5

Table C.10: Cuff options

	Nothing	*Collar*	*Hood*	*Band*
Nothing	5	1	1	1
Collar	1	5	2	3
Hood	1	2	5	2
Band	1	3	2	5

Table C.11: Neck options

Appendix D

Example Sketches

Table D.1 lists all roles in a garment, with the exception of those in the collar or hood. Where there is a cross ('X') in the cell, that role is featured in the sketch of the appropriate figure. Figures D.1 and D.2 show the front of a garment, whereas figures D.4 and D.5 show a sleeve. The sleeves are depicted with a vertical axis of symmetry for presentation purposes: SEACOP shows them with an orientation perpendicular to this. Collars and hoods are not shown as these pieces are deliberately created with very few constraints, leaving the user to edit the shapes to suit their requirements.

	D.1	D.2	D.3	D.4	D.5
ARMHOLE_BOTTOM	X	X	X		
ARMHOLE_TOP	X	X	X		
ARM_CURVE	X				
BAND			X		
BOTTOM_BORDER_LEFT	X	X			
BOTTOM_BORDER_RIGHT	X	X			
BOTTOM_LEFT	X	X	X		
BOTTOM_OF_DIVIDE		X			
BOTTOM_OF_SHAPING				X	X
CROWN				X	X
CUFF_EDGE				X	
CUFF_MIDDLE				X	
CURVE_EDGE				X	X
FITTED_OR_BAGGY_WAIST	X	X			
FRONT_BORDER_BOTTOM		X			
FRONT_BORDER_TOP		X			
LEFT_EDGE				X	X
NECK_BAND_LEFT			X		
NECK_BAND_RIGHT			X		
NECK_BEZIER_1	X				
NECK_BEZIER_2	X				
NECK_BOTTOM	X	X	X		
NECK_TOP	X	X	X		
ORIGIN	X	X	X	X	X
POCKET_BOTTOM_LEFT			X		
POCKET_BOTTOM_RIGHT			X		
POCKET_CENTRE			X		
POCKET_TOP_LEFT			X		
POCKET_TOP_RIGHT			X		
RAGLAN_BODY		X	X		
RAGLAN_BOTTOM				X	
RAGLAN_TOP				X	
RECT_PANEL_1				X	
RECT_PANEL_2				X	
RECT_PANEL_3				X	
RECT_PANEL_4				X	
SLEEVE_BEZIER_1					X
SLEEVE_BEZIER_2					X
STANDARD_BUTTON		X			
TOP_OF_SHAPING				X	X
USER_ADDED				X	
YOKE_BORDER_LEFT	X				
YOKE_BORDER_RIGHT	X				

Table D.1: Roles

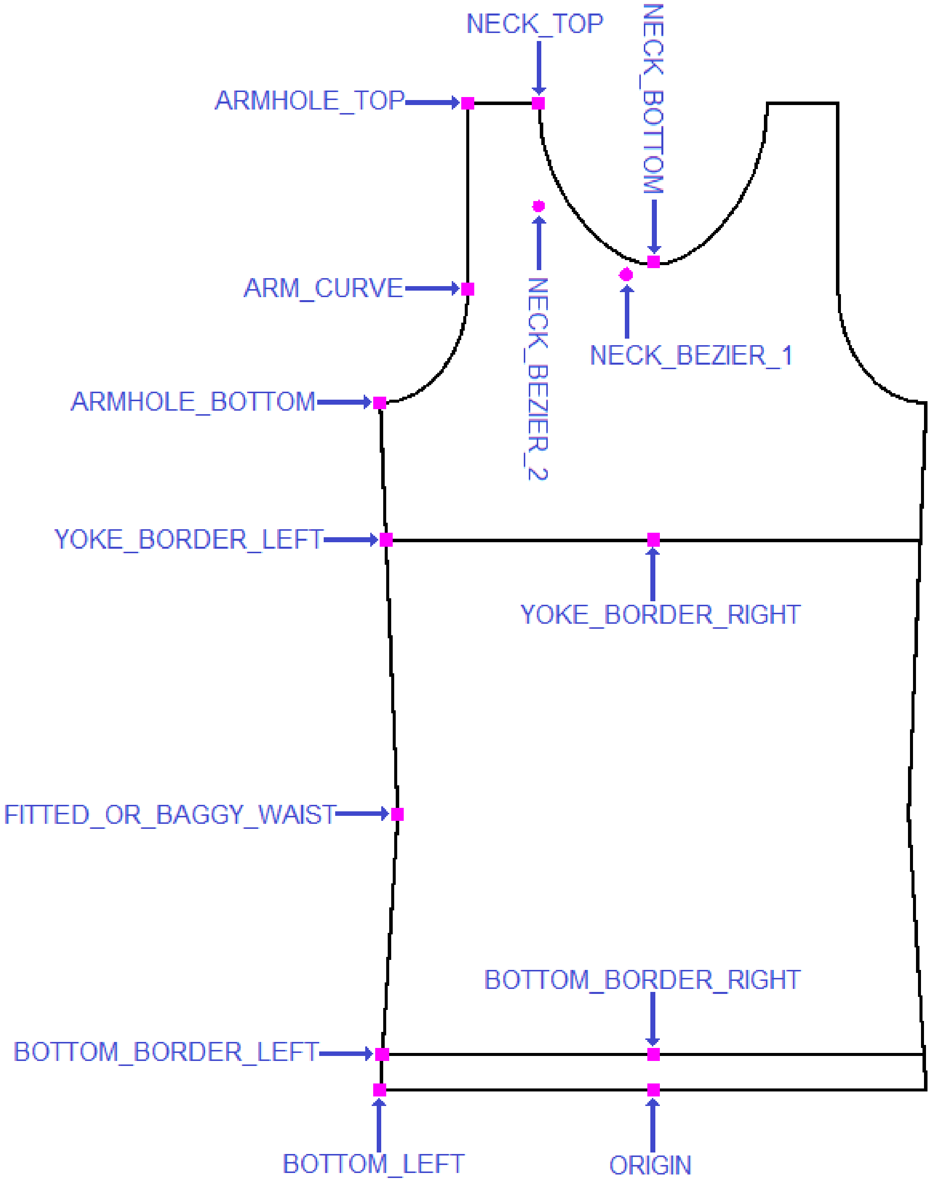


Figure D.1: Front of a round-neck sweater with a yoke and bottom border and a fitted waist

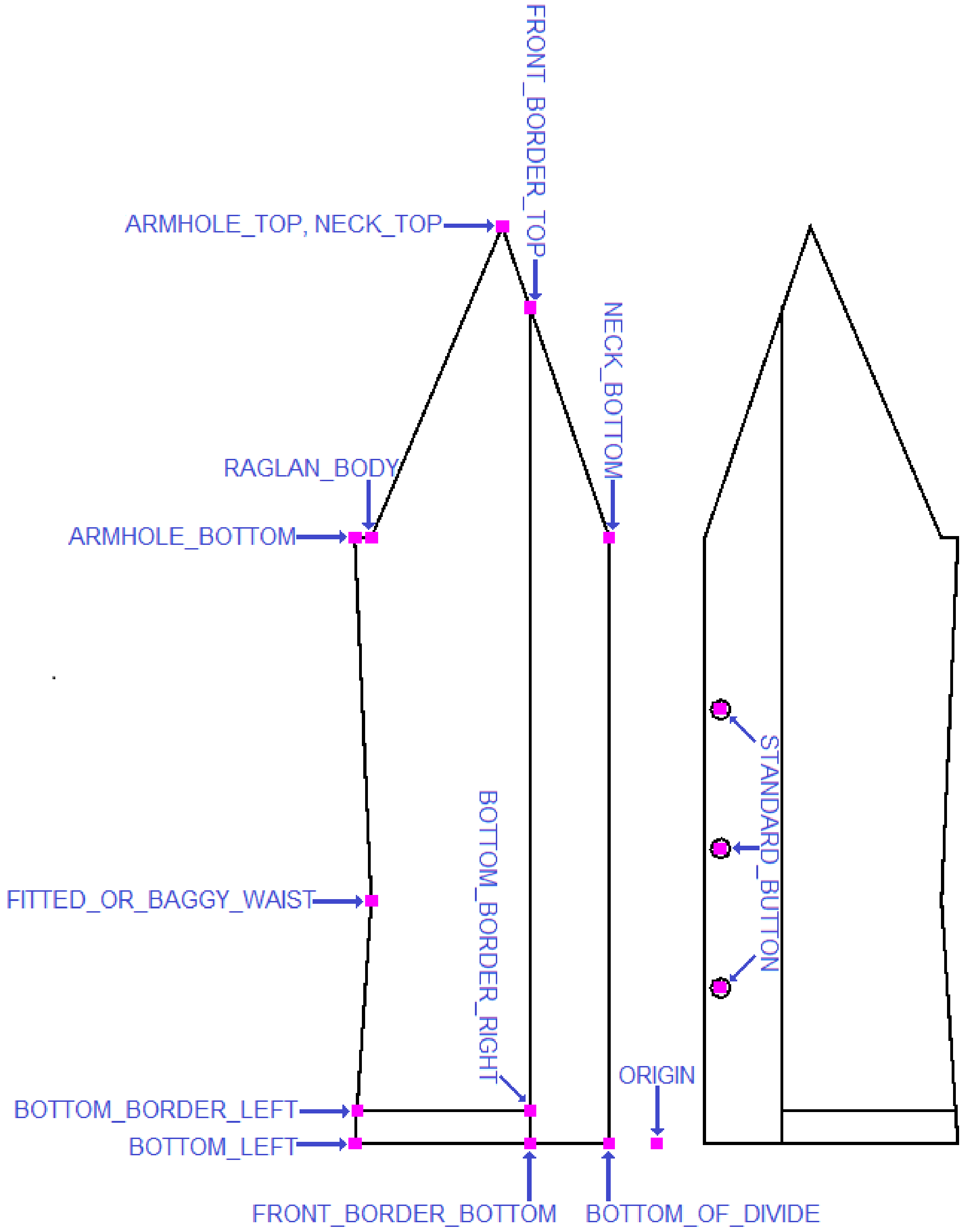


Figure D.2: Front of a v-neck cardigan with a yoke and a bottom border and a fitted waist (3 buttons)

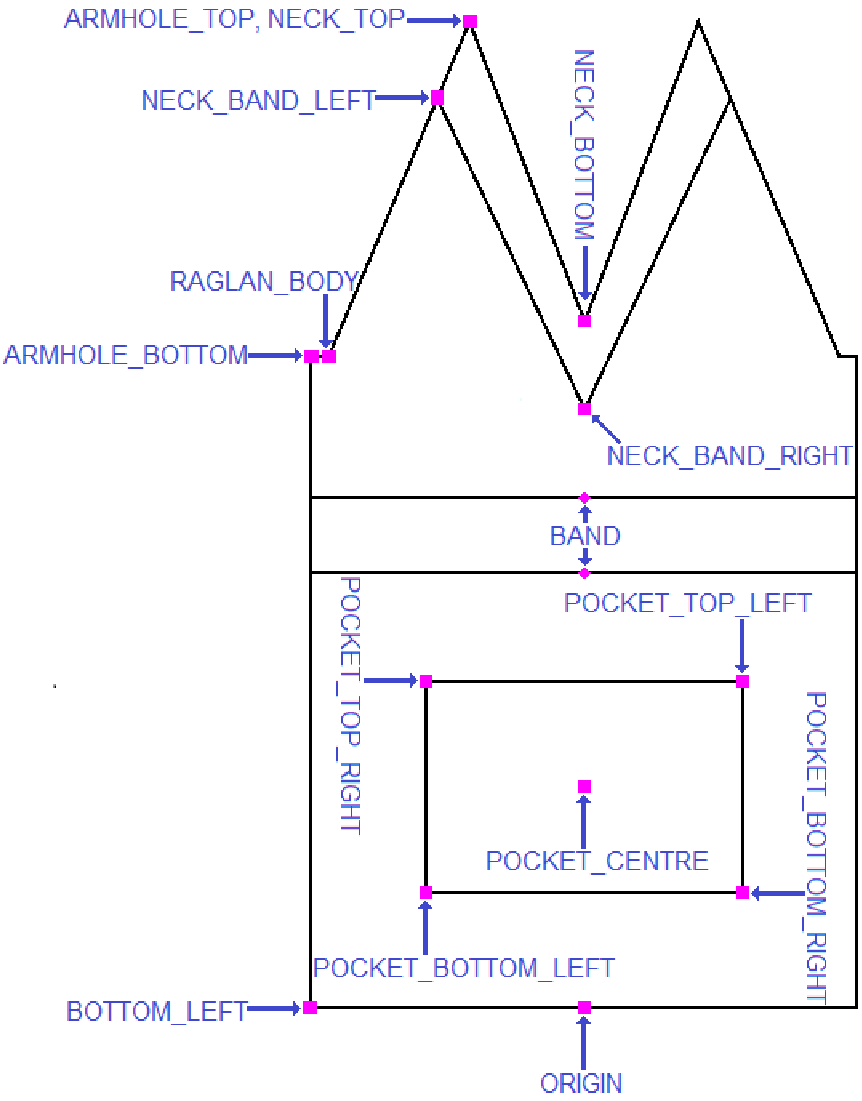


Figure D.3: v-neck Raglan sweater with a large centre pocket and horizontal band

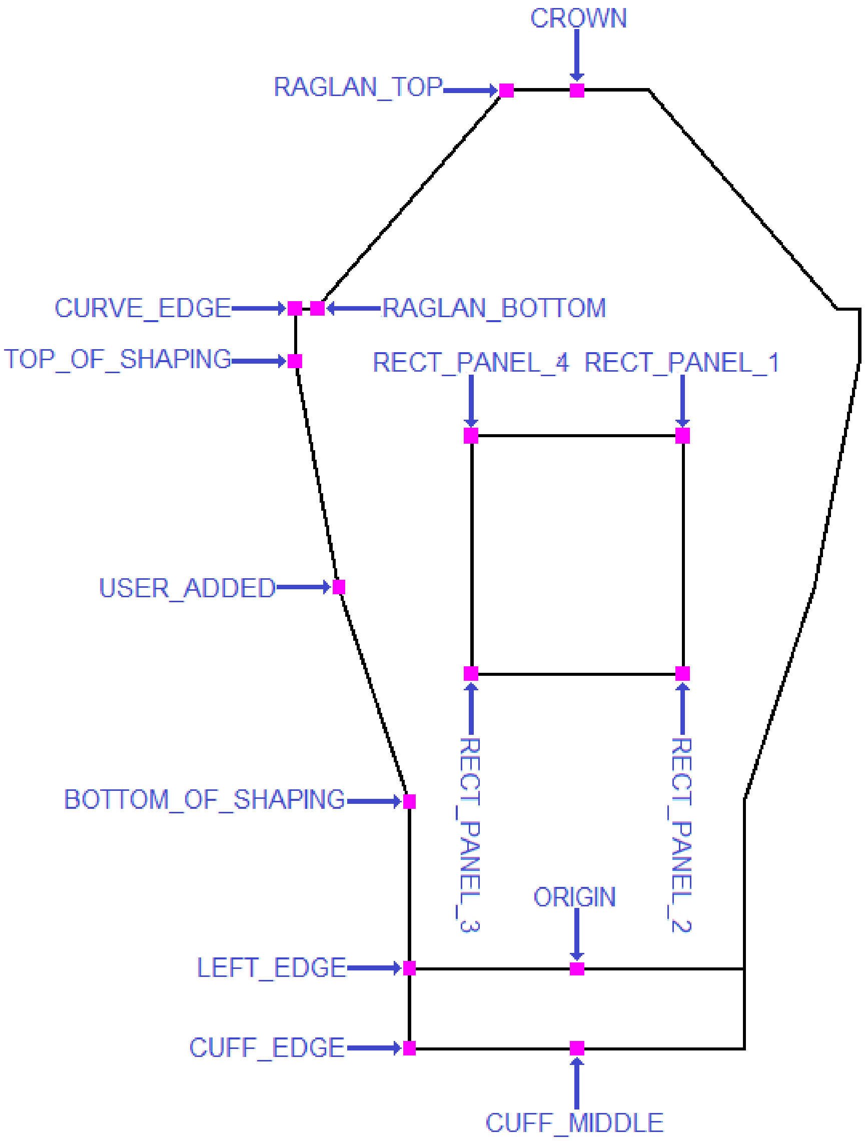


Figure D.4: Raglan sleeve with rectangular panel

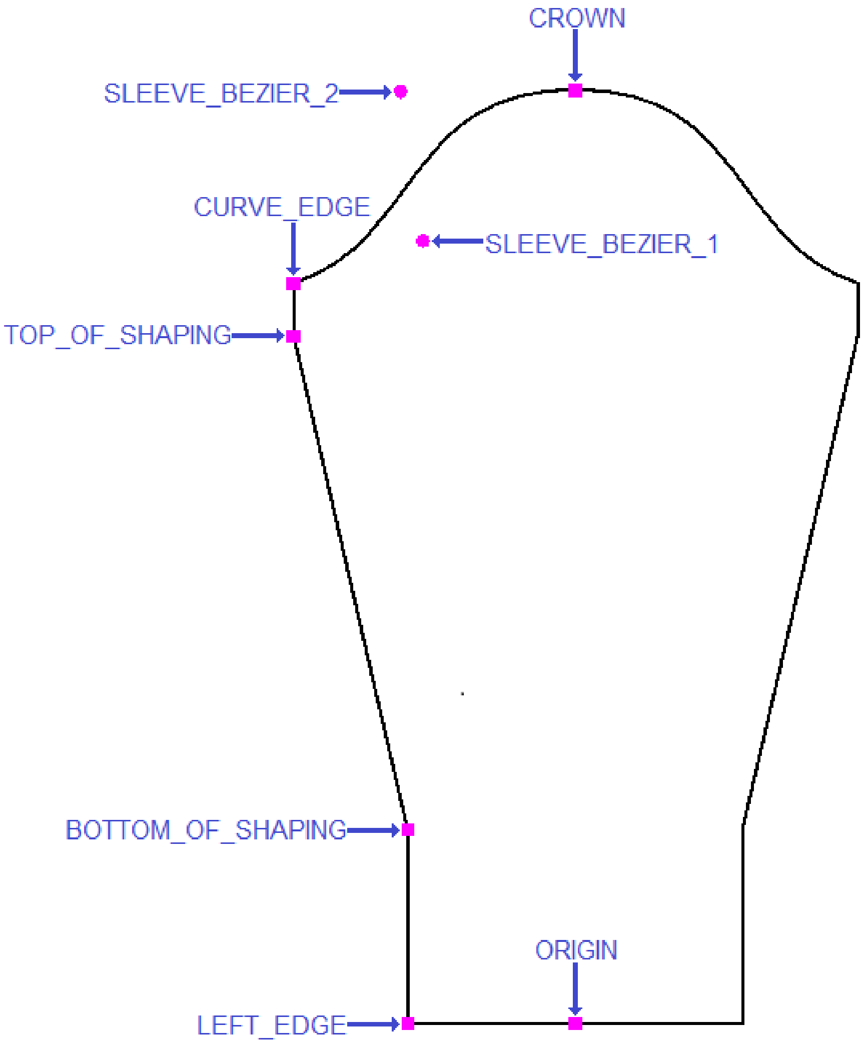


Figure D.5: Set-in sleeve

Appendix E

Case Base Composition

In table E.1, the prefix of the name indicates the source. Badger[6] and Budd [174] are knitting books. The remaining designs (prefixed 'PR') were created by myself. Table E.2 consists solely of cases supplied by Sirdar Spinning Ltd. For brevity, only the most important questionnaire attributes are shown in the table.

Name	*Type*	*Fastener*	*Has front border?*	*Waist*	*Has bottom border?*	*Yoke ?*	*Neck*	*Neck Option*	*Pockets*	*Armhole*	*Sleeve straights*	*Sleeves*	*Cuffs*
badger123	cardigan	buttons	yes	normal	yes	no	V	band	yes	semi set-in	n/a	n/a	n/a
badger123a	cardigan	buttons	yes	normal	yes	no	V	band	yes	semi set-in	n/a	n/a	n/a
badger128	sweater	n/a	n/a	normal	yes	no	round	collar	none	set-in	top	wrist	normal
badger128a	sweater	n/a	n/a	normal	yes	no	round	collar	none	dropped	top	wrist	normal
budd30	sweater	n/a	n/a	normal	no	no	round	band	none	dropped	bottom	wrist	none
budd30a	sweater	n/a	n/a	normal	no	no	V	band	none	dropped	bottom	wrist	none
budd30b	sweater	n/a	n/a	normal	no	no	round	band	none	dropped	bottom	wrist	none
budd30c	sweater	n/a	n/a	normal	no	no	V	band	none	dropped	bottom	wrist	none
budd58	sweater	n/a	n/a	normal	no	no	round	band	none	semi set-in	both	wrist	none
budd58a	sweater	n/a	n/a	normal	no	no	V	band	none	semi set-in	both	wrist	none
budd58b	sweater	n/a	n/a	normal	no	no	round	band	none	semi set-in	both	wrist	none
budd58c	sweater	n/a	n/a	normal	no	no	V	band	none	semi set-in	both	wrist	none
pr1	sweater	n/a	n/a	baggy	no	yes	slash	collar	yes	set-in	top	above elbow	none
pr10	sweater	n/a	n/a	normal	no	no	scoop	nothing	none	set-in	both	wrist	normal
pr11	cardigan	bother	no	normal	no	yes	V	nothing	none	Raglan	both	wrist	normal
pr12	sweater	n/a	n/a	normal	yes	no	slash	band	none	semi set-in	both	wrist	normal
pr2	cardigan	bother	no	baggy	no	no	scoop	nothing	yes	dropped	n/a	n/a	n/a
pr3	cardigan	buttons	yes	baggy	yes	no	straight	collar	none	set-in	both	above elbow	loose
pr4	sweater	n/a	n/a	fitted	no	yes	slash	nothing	yes	set-in	top	$\frac{3}{4}$ length	normal
pr5	cardigan	none	yes	normal	yes	no	round	nothing	none	set-in	n/a	n/a	n/a
pr6	sweater	n/a	n/a	fitted	yes	yes	round	collar	none	set-in	n/a	n/a	n/a
pr7	sweater	n/a	n/a	baggy	no	no	scoop	nothing	none	set-in	both	$\frac{3}{4}$ length	loose
pr8	cardigan	buttons	yes	normal	yes	yes	slash	collar	none	dropped	both	wrist	normal
pr9	sweater	n/a	n/a	baggy	yes	no	V	nothing	none	Raglan	top	wrist	loose

Table E.1: Miscellaneous cases

Name	*Type*	*Fastener*	*Has front border?*	*Waist*	*Has bottom border?*	*Yoke ?*	*Neck*	*Neck Option*	*Pockets*	*Armhole*	*Sleeve straights*	*Sleeves*	*Cuffs*
s2675	sweater	n/a	n/a	normal	no	no	scoop	nothing	none	dropped	both	wrist	normal
s2678	cardigan	none	yes	normal	yes	no	round	collar	none	dropped	both	wrist	loose
s2930	sweater	n/a	n/a	normal	yes	no	round	band	none	dropped	none	wrist	normal
s2967	cardigan	bother	yes	normal	yes	no	straight	nothing	none	semi set-in	both	wrist	loose
s2980	cardigan	none	yes	normal	yes	no	V	band	none	semi set-in	bottom	wrist	normal
s2999	cardigan	buttons	yes	normal	yes	no	V	band	yes	Raglan	none	wrist	normal
s3003	sweater	n/a	n/a	normal	yes	no	straight	band	none	semi set-in	none	very short	none
s3011	sweater	n/a	n/a	fitted	no	yes	straight	nothing	none	semi set-in	n/a	n/a	n/a
s3011a	sweater	n/a	n/a	fitted	no	yes	straight	nothing	none	semi set-in	bottom	$\frac{3}{4}$ length	normal
s3014	sweater	n/a	n/a	normal	no	no	scoop	band	none	semi set-in	none	very short	none
s3014a	sweater	n/a	n/a	normal	no	no	scoop	band	none	semi set-in	n/a	n/a	n/a
s3015	cardigan	buttons	no	normal	no	no	straight	nothing	none	set-in	bottom	wrist	none
s3017	sweater	n/a	n/a	normal	yes	no	round	band	none	dropped	bottom	wrist	loose
s3021	sweater	n/a	n/a	fitted	yes	no	scoop	band	none	semi set-in	none	very short	normal
s3021a	sweater	n/a	n/a	fitted	yes	no	scoop	band	none	set-in	n/a	n/a	n/a
s3041	cardigan	buttons	yes	fitted	yes	no	V	band	none	Raglan	bottom	wrist	none
s3065	sweater	n/a	n/a	normal	yes	no	V	nothing	none	semi set-in	none	very short	normal
s3132	sweater	n/a	n/a	fitted	no	no	round	collar	none	semi set-in	bottom	wrist	none
s3166	cardigan	buttons	yes	normal	yes	yes	round	band	none	semi set-in	none	very short	normal
s3213	cardigan	buttons	yes	normal	yes	no	V	collar	none	Raglan	top	$\frac{3}{4}$ length	normal
s3213a	cardigan	buttons	yes	normal	yes	no	V	collar	none	Raglan	none	very short	normal
s3224	sweater	n/a	n/a	fitted	yes	no	scoop	band	none	set-in	none	$\frac{3}{4}$ length	loose
s3224a	sweater	n/a	n/a	fitted	yes	no	scoop	band	none	set-in	none	wrist	loose
s3225	cardigan	buttons	yes	normal	no	no	straight	nothing	none	semi set-in	bottom	$\frac{3}{4}$ length	none
s3225a	cardigan	buttons	yes	normal	no	no	straight	nothing	none	semi set-in	none	very short	none
s3311	cardigan	buttons	yes	normal	yes	no	V	band	none	Raglan	none	wrist	normal

Table E.2: Cases from Sirdar Spining Ltd

Bibliography

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1994.

[2] Klaus-Dieter Althoff, Eric Auriol, Ralph Barletta, and Michel Manago. *Review of Industrial Case-Based Reasoning Tools*. AI Intelligence, 1995.

[3] Ralph Bergmann, Janet Kolodner, and Enric Plaza. Representation in case-based reasoning. *The Knowledge Engineering Review,*, 20(3):209–213, 2005.

[4] David B. Leake. *CBR in Context: The Present and Future (in Case-Based Reasoning: Experiences, Lessons, and Future Directions)*, chapter 1. AAAI Press/MIT Press, 1996.

[5] Jim Prentzas and Ioannis Hatzilygeroudis. Categorizing approaches combining rule-based and case-based reasoning. *Expert Systems*, 24(2):97–122, 2007.

[6] Ros Badger. *Knitting (Instant Expert)*. MQ Publications Limited, 2005.

[7] Roger C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, 1983.

[8] Mehmet Göker, Thomas Roth-Berghofer, Ralph Bergmann, Thomas Pantleon, Ralph Traphöner, Stefan Wess, and Wolfgang Wilke. The development of HOMER: A case-based CAD/CAM help-desk support tool. In B Smyth and P Cunningham, editors, *Advances in Case-Based Reasoning, Proceedings of the Fourth European Workshop on Case-Based Reasoning EWCBR98*, 1998.

[9] Ian Watson and Farhi Marir. Case-based reasoning: A review. *The Knowledge Engineering Review*, 9(4):335–381, 1994.

[10] Janet L Kolodner. Maintaining organization in a dynamic long-term memory. *Cognitive Science*, 7:243–280, 1983.

[11] Ralph Bergmann, Wolfgang Wilke, Ivo Vollrath, and Stefan Wess. Integrating general knowledge with object-oriented case representation and reasoning. In *Fourth German Workshop: Case-Based Reasoning-System Development and Evaluation*, 1996.

[12] Wolfgang Wilke and Ralph Bergmann. Techniques and knowledge used for adaptation during case-based problem solving. In *11th International Conference on Industrial and Engineering Applications of Artificial In telligence and Expert Systems: Tasks and Methods in Applied Artificial Intelligence*, pages 497–506, 1998.

[13] Gavin Finnie and Zhaohao Sun. R^5 model for case-based reasoning. *Knowledge-Based Systems*, 16(1):59–65, 2003.

[14] Paul Richards and A Ekárt. Hierarchical case based reasoning to support knitwear design. *CIRP Journal of Manufacturing Science and Technology*, 2(4):299–309, 2010.

[15] David Wilson. CBR Noir. In *AIII08*, 2008. Retrieved from http://www.aivideo.org/2008/accepted-videos.html, 6 February 2011.

[16] L Wittgenstein. *Philosophical investigations*. Blackwell, 1953.

[17] Janet. L. Kolodner and Robert L. Simpson. The mediator: Analysis of an early case-based problem solver. *Cogni*, 13(4):507–549, 1989.

[18] William M. Bain. A case-based reasoning system for subjective assessment. In *Proceedings of AIII08*, pages 523–527, 1986.

[19] Kristian J. Hammond. Chef: A model of case-based planning. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, 1986.

[20] Phyllis Koton. *Using Experience in Learning and Problem Solving*. PhD thesis, Massachusetts Institute of Technology, 1988.

[21] Phyllis Koton. A medical reasoning program that improves with experience. *Computer Methods and Programs in Biomedicine*, 30(2-3):177–184, 1989.

[22] E. Ray Bareiss, Bruce W Porter, and Craig C. Wier. Protos: an exemplar-based learning apprentice. *International Journal of Man-Machine Studies*, 29(5):549–561, 1988.

[23] David B. Leake. Creativity by case-based reasoning (CBR): Swale project home page. Accessed from http://www.cs.indiana.edu/ leake/projects/swale/ on 6 February 2011.

[24] Roger C. Schank and David B. Leake. Creativity and learning in a case-based explainer. *Artificial Intelligence*, 40(1-3):353–385, 1989.

[25] Pete Tierney. Developing a strategic platform for searching and retrieving corporate knowledge. Technical report, Inference Corporation, 1995. retrieved from http://www.riskinfo.com/tech/wpfull1.htm, 12 February 2011.

[26] Bradley P. Allen and S. Daniel Lee. A knowledge-based environment for the development of software parts composition systems. In *Proc. 11th ICSE*, pages 104–112, 1989.

[27] S.Daniel Lee, Trung D Nguyen, and Mary P. Czerwinski. Integration of case-based search engine into help database (Inference Corporation patent 5701399), 1997.

[28] Problem-solving software. Technical Report 20020083260, NASA, 1992.

[29] William Cheetham and John Graf. Case-based reasoning in color matching. In *ICCBR '97 Proceedings of the Second International Conference on Case-Based Reasoning Research and Development*, pages 1–12, 1997.

[30] William Cheetham. Tenth anniversary of the plastics color formulation tool. *AI Magazine*, 26(3):51–61, 2005.

[31] Xijun Wang. A web-based case-based reasoning tool. Master's thesis, University of Wyoming, 2000.

[32] Cynthia Marling, Mohammed Sqalli, Edwina Rissland, Hector Muñoz-Avila, and David Aha. Case-based reasoning integrations. *AI Magazine*, 23(1):69–86, 2002.

[33] Andrew M. Dearden and Derek G. Bridge. Choosing a knowledge based system to support a help desk. *Knowledge Engineering Review*, 8(3):201–222, 1993.

[34] R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *2nd International Joint Conference on Artificial Intelligence*, pages 608–620, 1971.

[35] Dana Nau. Automated planning: Theory and practice. Retrieved from http://www.cs.umd.edu/ nau/planning/slides, 25 February 2011.

[36] H Munoz-Avila and M.T. Cox. Case-based plan adaptation: An analysis and review. *Intelligent Systems*, 23(4):75 – 81, 2008.

[37] Santi Ontañón, Kane Bonnette, Praful Mahindrakar, Marco Gómez-Martin, Katie Long, Jai Radhakrishnan, Rushabh Shah, and Ashwin Ram. Learning from human demonstrations for real-time case-based planning. In *Proceedings of the IJCAI-09 Workshop on Learning Structural Knowledge from Observations*, 2009.

[38] Bernhard Nebel and Jana Koehler. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence*, 76:427–454, 1995.

[39] Claudia Eckert. The communication bottleneck in knitwear design: Analysis and computing solutions. *Computer Supported Cooperative Work*, 10:29–74, 2001.

[40] Janet L. Kolodner. Improving human decision making through case-based decision aiding. *AI Magazine*, 12(2):52–68, 1991.

[41] I. Watson. Case-based reasoning is a methodology not a technology. *Knowledge-Based Systems*, 12:303–308, 1999.

[42] Marvin Minsky. A framework for representing knowledge. In P.H. Winston, editor, *The Psychology of Computer Vision*, pages 211–217. McGraw-Hill, 1975.

[43] Sylvie Salotti and Véronique Ventos. Study and formalization of a case-based reasoning system using a description logic. In *EWCBR '98 Proceedings of the 4th European Workshop on Advances in Case-Based Reasoning*, 1998.

[44] Barry Smyth, Mark T. Keane, and Pádraig Cunningham. Hierarchical case-based reasoning integrating case-based and decompositional problem-solving techniques for plant-control software design. *IEEE Transactions on Knowledge and Data Engineering*, 13(5):793 – 812, 2001.

[45] Michael Redmond. Distributed cases for case-based reasoning; facilitating use of multiple cases. In *AAAI-90 Proceedings*, pages 304–309, 1990.

[46] D. Navinchandra, K. P. Sycara, and S. Narasimhan. Behavioral synthesis in CADET, a case-based design tool. In *Proceedings., Seventh IEEE Conference on Artificial Intelligence Applications, 1991*, pages 217 – 221, 1991.

[47] Bassant Mohamed El-Bagoury, Abdel-Badeeh Salem, and Hans-Dieter Burkhard. Hierarchical case-based reasoning behavior control for humanoid robot. *Annals of the University of Craiova - Mathematics and Computer Science Series*, 36(2):131–140, 2009.

[48] Andrea Bonzano and Pádraig Cunningham. Hierarchical CBR for multiple aircraft conflict resolution in air traffic control. In *Proceedings of 13th European Conference on Artificial Intelligence*, pages 58–62, 1998.

[49] L.Karl Branting and David W. Aha. Stratifed case-based reasoning: Reusing hierarchical problem solving episodes. In *IJCAI-95*, pages 384–390, 1995.

[50] Fong-Ching Yuan and Chaochang Chiu. A hierarchical design of case-based reasoning in the balanced scorecard application. *Expert Systems with Applications*, 36(1):333–342, 2009.

[51] Rainer Schmidt, Stefania Montani, Riccardo Bellazzi, Luigi Portinale, and Lothar Gierl. Case-based reasoning for medical knowledge-based systems. *International Journal of Medical Informatics*, 64:355–367, 2001.

[52] Brian Knight, Miltos Petridis, and Fei Ling Woon. Case selection and interpolation in CBR retrieval. *Expert Update*, 10(1):31–38, 2010.

[53] David B. Leake and David C. Wilson. Combining CBR with interactive knowledge acquisition, manipulation and reuse. In *Proceedings of the Third International Conference on Case-Based Reasoning and Development*, 1999.

[54] Manuela M. Veloso and Jaime G. Carbonell. Derivational analogy in prodigy: Automating case acquisition, storage and utilization. *Machine Learning*, 10:249–278, 1993.

[55] Mykola Galushka and David Patterson. Intelligent index selection for case-based reasoning. *Knowledge-Based Systems*, 19:625–638, 2006.

[56] Barry Smyth and Pádraig Cunningham. The utility problem analysed a case-based reasoning perspective. In *Proceedings of the Third European Workshop on Case-Based Reasoning*, pages 392–399, 1996.

[57] Janet Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers Inc, 1993.

[58] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[59] Pádraig Cunningham. A taxonomy of similarity mechanisms for case-based reasoning. Technical report, University College Dublin, 2008.

[60] John Kelleher and Brian Mac Namee. Instance-based learning (2). retrieved from http://www.comp.dit.ie/bmacnamee/materials/ml/lectures/Instance-Based-Learning-2.pdf.

[61] Gang Qian, Shamik Sural, Yuelong Gu, and Sakti Pramanik. Similarity between Euclidean and cosine angle distance for nearest neighbor queries. In *Proceedings of 2004 ACM Symposium on Applied Computing*, 2004.

[62] Jörg Walter Schaaf. Fish and shrink. a next step towards efficient case retrieval in large scaled case bases. In *EWCBR '96 Proceedings of the Third European Workshop on Advances in Case-Based Reasoning*, 1996.

[63] Amos Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.

[64] R Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[65] Dietrich Wettschereck and David W. Aha. Weighting features. In *Proceedings of the First International Conference on Case-Based Reasoning.*, 1995.

[66] Ramon Lopez De Mantaras, David McSherry, Derek Bridge, David Leake, Barry Smyth, Susan Craw, Boi Faltings, Mary Lou Maher, Michael T. Cox, Kenneth Forbus, Mark Keane, Agnar Aamodt, and Ian Watson. Retrieval, reuse, revision and retention in case-based reasoning. *Knowledge Engineering Review*, 20(3):215–240, 2005.

[67] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 20(5):99–121, 2000.

[68] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1965.

[69] Miltos Petridis, Soran Saeed, and Brian Knight. An automatic case based reasoning system using similarity measures between 3D shapes to assist in the design of metal castings. *Expert Update*, 10(2):43–51, 2010.

[70] B.T. Messmer and H. Bunke. Subgraph isomorphism in polynomial time. Technical report, University of Bern, 1995.

[71] T.A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, 1984.

[72] Xin Chen, Brent Francia, Ming Li, Brian McKinnon, and Amit Seker. Shared information and program plagiarism detection. *IEEE Transactions on Information Theory*, 50(7):1545–1551, 2004.

[73] Andrej Bratko, Gordon V. Cormack, Bogdan Filipic, Thomas R. Lynam, and Blaz Zupan. Spam filtering using statistical data compression models. *Journal of Machine Learning Research*, 7:2673–2698, 2006.

[74] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[75] Tor Gunnar Houeland. An efficient random decision tree algorithm for case-based reasoning systems. In *Twenty-Fourth International FLAIRS Conference*, pages 401–406, 2011.

[76] Brian Sheppard. World-championship-caliber scrabble. *Artificial Intelligence*, 134(1-2):241–275, 2002.

[77] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[78] Barry Smyth and Elizabeth McKenna. Footprint-based retrieval. In *Proceedings of the Third International Conference on Case-Based Reasoning and Development (ICCBR '99)*, pages 343–357, 1999.

[79] Mario Lenz, Hans-Dieter Burkhard, and Sven Brückner. Applying case retrieval nets to diagnostic tasks in technical domains. In *In Proceedings of the Third European Workshop on Case-Based Reasoning*, pages 219–233, 1996.

[80] Evangelos Simoudis and James Miller. Validated retrieval in case-based reasoning. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 310–315, 1990.

[81] Katy Börner. Structural similarity as guidance in case-based design. In *Proceedings of the First European Workshop on Case-Based Reasoning*, pages 197–208, 1993.

[82] Barry Smyth and Mark T. Keane. Adaptation-guided retrieval: questioning the similarity assumption in reasoning. *Artificial Intelligence*, 102:249–293, 1998.

[83] Sarah Jane Delany, Pádraig Cunningham, and Lorcan Coyle. An assessment of case-based reasoning for spam filtering. *Artificial Intelligence Review*, 24(3-4):359–378, 2005.

[84] Andres Gomez de Silva Garza and Mary Lou Maher. Using evolutionary methods for design case adaptation. In W. Jabi, editor, *Reinventing the Discourse - How Digital Tools Help Bridge and Transform Research, Education and Practice in Architecture, Proceedings of the Twenty First Annual Conference of the Association for Computer-Aided Design in Architecture*, pages 180–191, 2001.

[85] Lisa Purvis and Pearl Pu. Adaptation using constraint satisfaction techniques. In *Proceedings of the First International Conference on Case Based Reasoning*, pages 289–300, 1995.

[86] Susan Craw, Nirmalie Wiratunga, and Ray C. Rowe. Learning adaptation knowledge to improve case-based reasoning. *Artificial Intelligence*, 170:1175–1192 1175–1192, 2006.

[87] David McSherry. Demand-driven discovery of adaptation knowledge. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 222–227, 1999.

[88] David B. Leake, Andrew Kinley, and David Wilson. Acquiring case adaptation knowledge: A hybrid approach. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 684–689, 1996.

[89] Katia P. Sycara and D. Navinchandra. Index transformation techniques for facilitating creative use of multiple cases. In *Proceedings of the 12th International Joint Conference on AI*, pages 347–352, 1991.

[90] Mary Lou Maher and Dong Mei Zhang. CADSYN: A case-based design process model. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing*, 7:97–110, 1993.

[91] Mary Lou Maher. Engineering design synthesis: A domain independent representation. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (AI EDAM)*, 1:207–213, 1987.

[92] Edmund K. Burke, Bart L. MacCarthy, Sanja Petrovic, and Rong Qu. Multiple-retrieval case-based reasoning for course timetabling problems. *Journal of Operations Research Society*, 57(2):148–162, 2005.

[93] Manuela M. Veloso. Merge strategies for multiple case plan replay. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference*, pages 413–424, 1997.

[94] Michael E. Helms, Swaroop Vattam, and Ashok Goel. Compound analogical design, or how to make a surfboard disappear. In *30th Annual Meeting of the Cognitive Science Society*, pages 781–786, 2008.

[95] Elena I Teodorescu and Miltos Petridis. An architecture for multiple heterogeneous case-based reasoning employing agent technologies. In *1st International Workshop on Combinations of Intelligent Methods and Applications (CIMA 2008)*, pages 65–68, 2008.

[96] Daniel Hennessy and David Hinkle. Applying case-based reasoning to autoclave loading. *IEEE Expert*, 7(5):21–26, 1992.

[97] P. Richards and A Ekárt. Supporting knitwear design using case-based reasoning. In *Proceedings of the 19th CIRP Design Conference - Competitive Design*, pages 388–395, 2009.

[98] Jerry Slocum and Dic Sonneveld. *The 15 Puzzle*. Slocum Puzzle Foundation, 2006.

[99] Aaron F. Archer. A modern treatment of the 15 puzzle. *American Mathematical Monthly*, 106:793–799, 1999.

[100] Pádraig Cunningham. CBR: Strengths and weaknesses. In *Proceedings of 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, 1998.

[101] Ian Watson. A case study of maintenance of a commercially fielded case-based reasoning system. *Computational Intelligencee, Vol. 17 No. 2: pp.*, 17(2):387–398, 2001.

[102] Barry Smyth and Mark T. Keane. Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In *IJCAI'95 Proceedings of the 14th international joint conference on Artificial intelligence*, volume 1, pages 377–383, 1995.

[103] David C Wilson and David B. Leake. Maintaining case based reasoners: Dimensions and directions. *Computational Intelligence*, 17(2):196–213, 2001.

[104] Tor Gunnar Houeland and Agnar Aamodt. The utility problem for lazy learners - towards a non-eager approach. In I Bichindaritz and S Montani, editors, *Case-Based Reasoning Research and Development*, pages 141–155, 2010.

[105] Donald E. Knuth. Structured programming with go to statements. *Computing Surveys*, 6(4):261–301, 1974.

[106] William A. Wulf. A case against the goto. In *Proceedings of the 25th National ACM Conference*, volume 2, pages 791–797, 1972.

[107] Juan José Bello-Tomás, Pedro A. González-Calero, and Belén Díaz-Agudo. JColibri: An object-oriented framework for building CBR systems. In P.A. González-Calero and P. Funk, editors, *ECCBR 2004*, pages 32–46, 2004.

[108] Belén Díaz-Agudo, Pedro A. González-Calero, Juan A. Recio-García, and Antonio A. Sánchez-Ruiz-Granados. Building CBR systems with jCOLIBRI. *Science of Computer Programming*, 69(1-3):68–75, 2007.

[109] Juan A. Recio-García, Belén Díaz-Agudo, and Pedro A. González-Calero. Boosting the performance of CBR applications with jCOLIBRI. In *21st IEEE International Conference on Tools with Artificial Intelligence*, 2009.

[110] Juan. A. Recio-García, Belén Díaz-Agudo, Pedro González-Calero, and Antonio Sánchez-Ruiz-Granados. Ontology based cbr with jCOLIBRI. In R. Ellis, T. Allen, and A. Tuson, editors, *Applications and Innovations in Intelligent Systems XIV. Proceedings of AI-2006, the Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 149–162, 2006.

[111] Armin Stahl and Thomas R. Roth-Berghofer. Rapid prototyping of CBR applications with the open source tool myCBR. In *Proceedings of ECCBR 2008*, 2008.

[112] Michel Jaczynski and Brigitte Trousse. An object-oriented framework for the design and the implementation of case-based reasoners. In *In 6Th German Workshop on Case-Based Reasoning*, pages 33–42, 1998.

[113] J. A. Recio-García, Derek Bridge, Belén Díaz-Agudo, and Pedro. A. González-Calero. CBR for CBR: A case-based template recommender system for building case-based systems. In *Advances in case-based reasoning: 9th European conference, ECCBR*, pages 459–473, 2008.

[114] Juan A. Recio-García and Belén Díaz-Agudo. An introductory user guide to jCOLIBRI 0.3. Technical report, Universidad Complutense de Madrid, Spain, 2004.

[115] Cindy Marling, Edwina Rissland, and Agnar Aamodt. Integrations with case-based reasoning. *The Knowledge Engineering Review*, 20:241–245, 2005.

[116] J. Sabater, J. L. Arcos, and R. López de Mántaras. Using rules to support case-based reasoning for harmonizing melodies. In *Papers from the 1998 AAAI Spring Symposium*, pages 147–151, 1998.

[117] Jody J. Daniels and Edwina L. Rissland. What you saw is what you want: Using cases to seed information retrieval. In *Proceedings of ICCBR 97*, pages 325–336, 1997.

[118] Ioannis Hatzilygeroudis and Jim Prentzas. Combinations of case-based reasoning with other intelligent methods. *International Journal of Hybrid Intelligent Systems*, 6(4):189–209, 2009.

[119] Pablo Gervás, Belén Díaz-Agudo, Federico Peinado, and Raquel Hervás. Story plot generation based on CBR. *Journal of Knowledge Based Systems*, 18:2–3, 2005.

[120] Belén Díaz-Agudo and Pedro A. González-Calero. A declarative similarity framework for knowledge intensive CBR. In *Int. Conf. on Case-Based Reasoning ICCBR-2001*, pages 158–172, 2001.

[121] Florentino Fdez-Riverola and Juan M. Corchado. FSfRT: Forecasting system for red tides. *Applied Artificial Intelligence*, 17(10):251–264, 2003.

[122] B.G. Farley and W.A. Clark. Simulation of self-organizing systems by digital computer. *Institute of Radio Engineers Transactions on Information Theory*, 4:76–84, 1954.

[123] Rare condition named after twins, October 2005. BBC News article retrieved from `http://news.bbc.co.uk/1/hi/wales/south_west/4335454.stm` on 13 July 2011.

[124] Bruce W. Porter, Ray Bareiss, and Robert Holte. Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence*, 45(1-2):229–263, 1990.

[125] John Stillwell. *Mathematics and Its History*. Springer, 2010.

[126] Martin Müller. Computer Go. *Artificial Intelligence*, 134:145–179, 2002.

[127] Linda Adams. Learning a new skill is easier said than done. Retrieved from `http://www.gordontraining.com/free-workplace-articles/learning-a-new-skill-is-easier-said-than-done`, 13 July 2011.

[128] Michael M. Richter. The knowledge contained in similarity measures. In *Invited talk, ICCBR-95*, 1995.

[129] Nigel Cross. Forty years of design research. *Design Research Quarterly*, 1(2):3–5, 2006.

[130] Nigel Cross. *Design Thinking*. Berg, 2011.

[131] Ann Heylighen, W. Mike Martin, and Humberto Cavallin. How to teach and archive tacit design knowledge. *Design Intelligence*, 11(6), 2005.

[132] W Ju, L Neeley, and L Leifer. Design, design & design; an overview of Stanford's center for design research. In *Position paper for Workshop on Exploring Design as a Research Activity, CHI 2007*, 2007.

[133] Autodesk website, http://usa.autodesk.com/company, retrieved 21st July 2012.

[134] Wei-Cheng Xie, Xiu-Fen Zou, Jian-Dong Yang, and Jie-Bin Yang. Iteration and optimization scheme for the reconstruction of 3D surfaces based on non-uniform rational b-splines. *Computer-Aided Design*, 44(11):1127–1140, 2012.

[135] P.E. Bézier. How Renault uses numerical control for car body design and tooling. Technical Report SAE Paper 680010, Society of Automotive Engineers, 1968.

[136] Yuki Kineria, Mingsi Wang, Hongwei Lin, and Takashi Maekawa. B-spline surface fitting by iterative geometric interpolation/approximation algorithms. *Computer-Aided Design*, 44(7):697–708, 2012.

[137] Dong-Ming Yan, Wenping Wang, Yang Liuc, and Zhouwang Yang. Variational mesh segmentation via quadric surface fitting. *Computer-Aided Design*, 44:1072–1082, 2012.

[138] Yong Tsui Lee and Fen Fang. A new hybrid method for 3D object recovery from 2D drawings and its validation against the cubic corner method and the optimisation-based method. *Computer-Aided Design*, 44(11):1090–1102, 2012.

[139] Alan Sullivan, Huseyin Erdim, Ronald N. Perry, and Sarah F. Frisken. High accuracy NC milling simulation using composite adaptively sampled distance fields. *Computer-Aided Design*, 44(6):522–536, 2012.

[140] W. Anotaipaiboon and S.S. Makhanov. Minimization of the kinematics error for five-axis machining. *Computer-Aided Design*, 43(12):1740–1757, 2011.

[141] Ashok K. Goel, Swaroop Vattama, Bryan Wiltgen, and Michael Helms. Cognitive, collaborative, conceptual and creative - four characteristics of the next generation of knowledge-based CAD systems: A study in biologically inspired design. *Computer-Aided Design*, 44(10):879–900, 2012.

[142] Yan Liang, Ying Liu, Chun Kit Kwong, and Wing Bun Lee. Learning the 'whys': Discovering design rationale using text mining - an algorithm perspective. *Computer-Aided Design*, 44(10):916–930, 2012.

[143] Claudia Eckert and Martin Stacey. CAD systems and the division of labour in knitwear design. In *Proceedings of the IFIP TC9/WG9.1 Fifth International Conference on Woman, Work and Computerization: Breaking Old Boundaries - Building New Forms*, pages 409–422, 1994.

[144] Claudia Eckert. Generic and specific process models: Lessons from modelling the knitwear design process. In *Proceedings of the Sixth International Symposium on Tools and Methods of Competitive Engineering (TMCE 2006)*, pages 681–692, 2006.

[145] Claudia Eckert. *Intelligent Support for Knitwear Design*. PhD thesis, The Open University, 1997.

[146] C.M. Eckert, M.K. Stacey, and P.J. Clarkson. Algorithms and inspirations: Creative reuse of design experience. In *Proceedings of the Greenwich 2000 Symposium: Digital Creativity.*, pages 1–10, 2000.

[147] A. Ekárt. Genetic programming for the design of lace knitting stitch patterns. In R. Ellis, T. Allen, and M. Petridis, editors, *Applications and Innovations in Intelligent Systems XV, Proceedings of AI-2007, the Twenty-seventh SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 261–274, 2007.

[148] Brendon Woodford. The DesignAdvisor: An intelligent knitwear design aid. Master's thesis, The University of Otago,, 1997.

[149] Joel Peterson, Jonas Larsson, and Rudrajeet Pal. Co-design tool for customised knitwear. In *AUTEX 2009 World Textile Conference*, 2009.

[150] Dictionary.com entry on "Raglan", http://dictionary.reference.com/browse/raglan, retrieved 24th July 2012.

[151] Personal conversation with Dr Claudia Eckert, 3rd March 2009.

[152] Paul Richards and Anikó Ekárt. Automating a knitwear design process using case-based reasoning. In *10th International Conference on The Modern Information Technology in the Innovation Processes of the Industrial Enterprises*, pages 390–395, 2008.

[153] Java API documentation, retrieved from http://docs.oracle.com/javase/7/docs/api on 14 July 2012.

[154] Jyrki Katakainen and Jesper Larsson Träff. A meticulous analysis of mergesort programs. In *Proceedings of the 3rd Italian Confrence on Algorithms and Complexity, Lecture Notes in Conputer Science 1203*, pages 217–228. Springer-Verlag, 1997.

[155] S Ramanujan. Modular equations and approximations to Pi. *Quarterly Journal of Mathematics*, 45:350–372, 1914.

[156] Shahram Zafary. A single term formula for approximating the circumference of an ellipse, 2009. http://mathforum.org/kb/servlet/JiveServlet/download/128-1921864-6683056-551607/circumference retrieved on 14th July 2012.

[157] M. Zeleny and FA von Hayek. Management support systems: Towards integrated knowledge management. *Human Systems Management*, 7(1):59–70, 1987.

[158] K. S. Lee and C. Luo. Application of case-based reasoning in die-casting die design. *International Journal of Advanced Manufacturing Technology*, 20:284–295, 2002.

[159] Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo*, 22:1–74, 1906.

[160] Jung-Eun Lee, Rong Jin, Anil K. Jain, and Wei Tong. Image retrieval in forensics: Tattoo image database application. *IEEE Multimedia in Forensics, Security, and Intelligence*, 19(1):40–49, 2012.

[161] Fazal Malik and Baharum Baharudin. Median and Laplacian filters based feature analysis for content based image retrieval using color histogram refinement method. *Journal of Applied Sciences*, 12(5):416–427, 2012.

[162] Facundo Mémoli and Guillermo Sapiro. Comparing point clouds. In *SGP '04 Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 32–40, 2004.

[163] Remco C. Veltkamp and Michiel Hagedoorn. State-of-the-art in shape matching. Technical report, Department of Computer Science, Utrecht University, 1999.

[164] Derek Justice and Alfred Hero. A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1200–1214, 2006.

[165] Eric C. McCreath. Partial matching of planar polygons under translation and rotation. In *Proceedings of the 20th Canadian Conference on Computational Geometry*, pages 47–50, 2008.

[166] Shlomo S. Sawilowsky. Invited debate: Target article you think you've got trivials? *Journal of Modern Applied Statistical Methods*, 2(1):218–225, 2003.

[167] S. Srinivasan, Jagjit Singh, and Vivek Kumar. Multi-agent based decision support system using data mining and case based reasoning. *International Journal of Computer Science Issues*, 8(4):340–349, 2011.

[168] Ibrahim Adepoju Adeyanju. *Case Reuse in Textual Case-Based Reasoning*. PhD thesis, Robert Gordon University, 2011.

[169] Fadi Badra, Amélie Cordier, and Jean Lieber. Opportunistic adaptation knowledge discovery. In *Proceedings of the 8th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, 2009.

[170] Thomas R. Hinrichs. *Problem Solving in Open Worlds: A Case Study in Design*. Lawrence Erlbaum Associates, 1992.

[171] T.-C. Au, H. Muñoz-Avila, and D. S. Nau. On the complexity of plan adaptation by derivational analogy in a universal classical planning framework. In *In European Conference on Case-Based Reasoning (ECCBR)*, pages 13–27, 2002.

[172] Douglas M. Hawkins. The problem of overfitting. *J. Chem. Inf. Comput. Sci.*, 44:1–12, 2004.

[173] David Leake and Mark Wilson. How many cases do you need? Assessing and predicting case-base coverage. In *Case-Based Reasoning Research and Development: Proceedings of the Nineteenth International Conference on Case-Based Reasoning, ICCBR-11*, pages 92–106, 2011.

[174] Ann Budd. *The Knitter's Handy Book of Sweater Patterns: Basic Designs in Multiple Sizes and Gauges*. Interweave Press LLC, 2004.

[175] Hershey H. Friedman and Taiwo Amoo. Rating the rating scales. *Journal of Marketing Management*, 9(3):114–123, 1999.

[176] Gene find casts doubt on double 'cot death' murders, John Sweeney and Bill Law, The Observer, 15 July, 2001.

Index