COMPUTER BASED APPROACHES TO MANAGERIAL DECISION MAKING: OPTIMISATION, HEURISTICS, AND SIMULATION

Ling LING

B.Sc. (Peking University),

M.Sc. by Research in Business Management

THE UNIVERSITY OF ASTON IN BIRMINGHAM

September, 1995.

This copy of the thesis has been supplied on condition that any one who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without proper acknowledgement.

THE UNIVERSITY OF ASTON IN BIRMINGHAM

COMPUTER BASED APPROACHES TO MANAGERIAL DECISION MAKING:

OPTIMISATION, HEURISTICS, AND SIMULATION

Ling Ling
M.Sc. by Research in Business Management

September, 1995

This research studies three computer based approaches - optimisation, heuristic, and simulation, to numerical optimisation problem in managerial decision making.

In Chapter 2 we first discuss computer based mathematical programming with the focus on linear programming (LP) and its software in the application of the simplex and interior-point methods. We study non-linear programming (NLP) and its software through unconstrained optimisation methods and constrained optimisation methods. Instead of discussing dynamic programming (DP) directly, we discuss an important large-scale optimisation technique - decomposition. A number of definitions for heuristics are discussed and clarified in Chapter 3. Substitution is proposed as a principle which can be used to classify heuristics. Since many well-known heuristics can be categorised as problem substitution heuristics, model substitution heuristics, or algorithm substitution heuristics. The heuristic approach is then studied according to these classifications. The concept of computer simulation is discussed in Chapter 4. The method to categorise computer simulation according to the major objectives: problem-finding, problem-understanding, and problem-solving. In order to demonstrate these approaches, a case study using different approaches is proposed in Chapter 5.

On the basis of the discussion on managerial decision making and an interview with the OR professionals, we carried out an extensive discussion on these three approaches through methodology consideration, modelling consideration, feasibility consideration, and applicability consideration.

This research indicates that heuristics are more suitable for managerial decision making in many situations because optimisation approach still needs to be improved for managerial decision making. It is concluded that operations management should consider integrating these heuristic methods into a decision support system. Simple, understandable and usable heuristic approaches for solving managerial decision making problems are needed. Heuristics will be more attractive to managers in the future with the development of IT. Such approaches will undoubtedly be built around computer-based decision support systems. Computer simulation may then be regarded as the last resort. Despite this, it is surprising how often such an approach is needed. There are certain advantages in employing a simulation approach in management science and it may be the only way of tackling some managerial decision making problems. Diversified heuristics will be widely applied in managerial decision making.

Key words: optimisation, heuristics, simulation, computer based approach, decision making.

Acknowledgements

I am most indebted to my supervisor, Mr J. B. Kidd who introduced and encouraged me to explore this interesting problem area, for his excellent guidance, constant help, and considerations throughout this research.

My acknowledgements are due to Dr. J. S. Edwards for his kind concern, constructive criticism, and precious advice. I would like to thank specially Mr. K. Rapley and his colleagues (Dr. Roger Blackburn, Dr. Himadri Chatterjee, and Ms Chand Akbar) at British Airways for their kind-hearted help and valuable information during my visit to Heathrow.

I am grateful to my colleague Mr. D. Pritchard for his excellent translation of an important paper from the German, as well as to Mr. S. L. Robinson for providing me with some precious literature.

I would like to thank Mr. C. Hart who greatly improved the quality of the English in this thesis. I will also remember many members of staff at Aston Business School, especially Mrs. P. E. Lewis who was always willing to help throughout my study.

Finally, I am deeply indebted to my family for their concern, encouragement, help, and love.

CONTENTS

List of Tables, Figures etc.

1. Introduction

1.1 Numerical optimisation problems in managerial decision making

The task of managers is to plan, organise, and control the activities of a business. To accomplish this, management is usually in a constant state of decision making. Generally, management makes such decisions in two ways: First, the manager's understanding and perception of the problems involved and their resolutions serve as the basis for choice. Second, mathematics, numerical methods, and simulation systems expressed in computer programs can assist in the decision process.

We do not wish to make an instant judgement of a particular problem, various modes of approach are possible. Firstly, it may be possible to conduct experiments directly on the real system or physical system. Second, he may be able to construct and use a mathematical model of the system of interest. A third possibility is to simulate the system through a PC or mainframe environment.

Referring to a managerial decision making problem, it can be mathematically classified into three groups:

1) Well-structured problem such as a mathematical programming based problem or other standard OR problems, where it has been proved that there exists an optimal solution.
2) Ill-structured problem such as some dynamic or stochastic problems, where it has been proved that there is not an optimal solution.
3) Unknown structured problem such as some complex system, where it has not been proved whether there is or not an optimal solution.

A managerial decision making problem may be modelled as any one of the above. An optimal algorithm for a well-structured problem is a procedure which is guaranteed to converge to an optimal solution at any given accuracy.

Numerical optimisation problems occur in all areas of managerial decision making, arising whenever there is a need to minimise (or maximise) an objective function that depends on a set of variables while satisfying some constraints if necessary.

1.2 Popular approaches

It is well-known that an optimisation algorithm will determine an optimum of a well-formulated numerical optimisation problem. By definition, there does not exist any other solution which gives a better value of the objective than that of the optimum. Attempting to improve the quality of managerial decisions, many OR/MS people continue to explore the opportunity to develop optimisation algorithms and their mathematical theory.

Basically, five ways exist for finding an optimum once a model has been developed:

1. The most commonly used method is the intuitive procedure.
2. The first most commonly used method for finding an optimum is differential calculus, a branch of classic mathematics.
3. A form of optimisation is known as mathematical programming or extremum-finding, e.g., linear programming, and non-linear programming etc.
4. A currently used method is the heuristic method, which is capable of formal presentation, but does not guarantee optimality.
5. A direct search in conjunction with computer simulation for the optimum becomes necessary when a model does not fulfil the requirements for either calculus or extremum-finding.

The first method is not capable of formal presentation and, therefore, will not be discussed further.

Assumptions on the second method must be made about the continuity or even differentiability of the process, that is, the process cannot have discrete changes of value.

These assumptions do not hold in many business situations. The techniques of calculus, apparently so powerful for studying the physical world, fail in many elementary business situations because of the difficulty in defining an adequate mathematical expression of the problem. So they will also not be discussed further.

Mathematical programming algorithms require that the model be formulated according to specific assumptions. As an example, the linear programming procedure is the prototype extremum-finding algorithm. Linear programming, which became significant with the discovery of the simplex algorithm, assures that the optimum will be found efficiently in linear allocation problems. Other extremum-finding procedures are available for inventories, equipment replacement, and some simple queuing situations.

Mathematicians discovered that the golden principle of optimisation: to check every feasible solution and to find the best, was wrong. This approach is far from being practical since the feasible solutions of a selection and arrangement problem would increase sharply with the size of the problem. In this situation, heuristics are the only way to solve a combinatorial optimisation problem. Therefore, it is not surprising that people choose heuristics when optimisation becomes impossible. The heuristic depends upon a concrete modelling situation. Simplification is the major principle whenever it is applied to an optimisation problem.

In the final method, the word 'search' should not be confused with search theory, which is a particular type of model dealing with situations in which one object searching for another.

Some managerial decision making problems fall into combinatorial optimisation where objects are discrete, such as sequencing, scheduling, routing, layout and design problems. Apart from the above five optimisation methods, a relatively new area of mathematics is termed *combinatorial optimisation*. It is often described as the selection and arrangement

of discrete objects. There is usually a finite number of feasible solutions to each instance of these problems. Therefore it seems intuitive that the fundamental theorem of combinatorial optimisation could be employed: examine every feasible solution and choose the best. Unfortunately, there are usually far too many solutions for this approach to be practical. In order to search for optimum decisions, sometimes solution procedure is purely intuitive; sometimes it is made under a set of guidelines or rules of thumb or heuristics; sometimes it is made by using more formal techniques such as hill climbing or random search.

Managers expect powerful tools for comparing alternative scenarios quantitatively, for making the effects of decisions visible and hence open to discussion, and for reducing uncertainty in complex situations. None of us should take the decision out of the manager's hands; we should help him to improve the quality of his decisions and to shorten the time to reach them. When it is necessary for managers to make decisions, Operational Research (OR) and Management Science (MS) can do more for them than many of the managers think. This is achieved by revealing the consequences of possible decisions in as quantitative a way as possible.

1.3 The nature of managerial decision making

After a decade of belt-tightening recession, intense cost pressure and fierce competition are driving companies to extract even more efficiency and productivity from managerial decision making, which becomes one of the decisive drivers of success or failure for companies of all sizes. In this section we briefly discuss the nature of managerial decision making.

Managers are often regarded as rational, purposeful, and decisive. However, we see them as going through a series of stages of analysis before deciding what to do.

The doing comes from the planning, planning comes from the thinking. Isenburg (1984) explored how senior managers think, he suggests two findings: First, it is hard to pinpoint if or when they actually make decisions about major business or organisational issues on their own. And second, they seldom think in ways that one might simply view as "rational," i.e., they rarely systematically formulate goals, assess their worth, evaluate the probabilities of alternative ways of reaching them, and choose the path that maximise expected return. Rather, managers frequently bypass rigorous, and analytical planning altogether, particularly when they face difficult, novel, or extremely entangled problems. When they do use analysis for a prolonged time, it is always in conjunction with intuition.

On what basis does the manager make his decision? Economists assumed complete rationality: The model was that of an economic man who deals with the real world in all its complexity, and who selects the rationally determined best course from all those available to him in order to maximise his return. It can be seen that classical theory tends to view a firm as an entrepreneur rather than as an organisation, assuming perfect knowledge of all market conditions, stress profit maximisation as the goal. It takes a firm to be an omnisciently rational system of business.

In place of an economic man Herbert Simon proposed a model of administrative man. While economic man selects the best course from those available to him, administrative man looks for a course of action that is satisfactory or good enough(Pugh, Hickson, Hingings, 1979).

In the place of an omnisciently rational system Richard Cyert and James March view a firm as an adaptively rational system, adapting and responding to a variety of internal and external constraints in arriving at decisions. Their behavioural theory of the firm is a notable effort to link classical economics theory to contemporary organisation theory. It is an attempt to describe and to explain how business decisions come to be made. Cyert and March take business firms as their starting point, and specifically have in mind the large

multi-product organisation operating under imperfect competition. The theory is about decisions such as what price to aim at, what volume to produce, and how resources are to be allocated within a firm. Decisions of these kinds are seen as choices, made in terms of objectives, from among a set of alternatives on the basis of whatever information is available(Pugh, Hickson, Hingings, 1979).

The three approaches to the numerical optimisation problem in managerial decision making can be regarded as method of selecting decision. Rivett (1994) proposed three basic elements in any decision:

1) The range of choice.
2) The consequences of each of these choices.
3) The objective(s) involved.

According to these elements, we will examine whether numerical optimisation plays an important role in managerial decision making.

In this research we suppose that a managerial decision making problem can be modelled well as a numerical optimisation problem in the cases where the range of choice is so limited, the consequence of each of them is so well determined and measured, and the objective is a single statement.

Referring to the range of choice, the modeller should have the knowledge that all the choices will be compared within the permitted time and available resources.

Referring to the consequences of each of these choices, all measurement involves a viewpoint of the modeller's in managerial decision making, for the units express what is thought to be important. Measures can be different, e.g., the cost consideration in the replenishment problem is expressed in cost per unit item, or cost per unit time, which deal with different objectives and are of interest to different people.

Referring to the objective(s) involved, there might be a number of conflicting objectives in a company. We often find many managers would like to accomplish more in less time. One of the implications is that when a manager addresses any particular problem, he calls a number of related problems to mind at the same time. For example, a sales manager may prefer better quality and their product line to be as full as possible with a large number of options but ignore the limited capacity and additional cost; a production manager may insist on the importance of synchronous manufacturing but ignore high inventory cost; a purchasing manager may emphasise the availability of raw materials and components but ignore financial difficulty and limited warehouse space; an inventory manager may attempt to reduce his inventory level but ignore current production capacity or favourable purchase prices. However, the managerial decision making here is much more concerned with single objective problem.

Any numerical optimisation problem should be represented by a mathematical model. In other words, model-building plays an important part in managerial decision making. Models offer insight and the possibility to compare decision scenarios with each other. Almost always the computer is an indispensable tool in today's management practice. A large part of the OR techniques can be used on the personal computers, software is becoming more and more user-friendly and cheaper. In recent years, there has been an enormous growth in the ease with which a problem area can be represented by a numerical model that the managers consider sufficiently realistic. Large quantities of data can easily be stored in databases that are simple to access. Operational research has developed powerful tools for comparing alternative scenarios quantitatively and for reducing uncertainty in complex situations.

When it is necessary for managers to make quantitative decisions, a numerical optimisation problem can help them more than many of them think. OR workers do not take decisions out of the manager's hands; but help him to improve the quality of the decisions and shorten the time to reach them. We also fully understand intuition,

experience and common-sense of the manager remain indispensable for the final two steps: the selection of a chosen solution and its implementation.

1.4 A story learnt from British Airways

In the previous section we discussed the nature of managerial decision making. Basically, most decisions are concerned not with searching for the sharpest needle in the haystack but with searching for a needle sharp enough to sew with. The development in management has been due to the application of such techniques as optimisation, heuristics, and computer simulation. Bearing this thought in mind, it is necessary to find out their characteristic, implications, and influence on managerial decision making.

At the initial stage of this research, I interviewed some key OR staff at Heathrow. The OR group at British Airways is a very active and successful team. They first admitted that they did do work for the engineering group but very little optimisation. During the interview, they kindly provided me with a set of papers, and plotted the history of one of their problems, which was most amenable to optimisation techniques.

The interesting managerial decision making problem is the so-called Ground Staff Rostering Schedule in British Airways: Basically the twenty four hours of the day are split into 15 minute periods, so each line up is 15 minutes. The assignment is the number of ground staff you need in that 15 minute period in order to complete the work. This is, ground staff checking people in for flights at terminal one at Heathrow. So you can see it peaks in the morning, right, ..., and peaks in the evening. The task is then to minimise the number of ground staff that you need in the day to actually work this roster. But you have got a different work load for different days of the week (i.e., the constraints vary with time). Furthermore, you also have got different ways of doing the rostering (i.e., the objective function varies with different service patterns).

So it is an integer program in that you need polynomials as input. You can formulate it as a linear program or you can solve it by using heuristics. I am very much more interested in which approach is most suitable.

It is said that four aspects of change in British Airways combined to influence the use of one of the popular optimisation techniques for this problem - Linear Programming.

1) **Changes in the business environment**

The increasing intensity of competition has put pressure on costs and efficiency, and produced an increase in the frequency and degree of schedule changes. An increasing realisation that manpower planning and rostering are both necessary and desirable has developed. This has been matched by a decrease in staff/union resistance to flexible rostering.

2) **Changes in the nature and ability of users**

Many of the changes were enabled and encouraged by OR's contribution. The story tells of users who were initially resistant to change and to computers becoming skilled roster analysts and model users; then partners in the development of methods and techniques, and finally highly skilled, computer literate, demanding users.

3) **Changes in information technology**

The introduction of a mainframe DEC 10 prefaced a successful approach to OR epitomised by heuristic modelling within a user-friendly, computerised decision support system. PCs raised user expectations even further making the DEC 10 user interface look primitive. LP packages have improved dramatically over the period of its introduction.

4) Changes in the practice of OR in British Airways

OR was changed by the growth and success of the DSS approach to OR, with its mistrust of black box solutions, and its reliance on user friendly DSS. In recent years, OR has been further changed by the loss of uniqueness and the leverage that DSS development has offered, as the I.T. professionals began to realise the power and need for a good DSS. Both of these points had some impact on the applicability of an optimisation approach such as LP.

It is also said that the OR group at British Airways has moved from a position of intellectual leadership to one of intellectual partnership (Graeme Davison, 1989). As a consequence, British Airways has achieved cost savings that run into many millions of pounds, while many other British industries have struggled and failed to reach first base in the quest to introduce flexible rosters.

On the basis of OR practice in British Airways, Graeme Davison (1989) described when they used linear programming and when they used heuristics and why. He described the on/off story of ground staff rostering in British Airways and the perhaps over-ambitious attempt to use the optimisation approach of LP or solve the workload cover and rostering problems in one step, How its black box image of LP was its downfall, leading to two separate heuristic models being used and how LP has recently re-emerged to solve a cut down version of the total requirement (i.e., covering problem), and how, alongside, some simple heuristics have been used.

Referring to the ground staff rostering problem, the expectation from the managers and limitation of the linear programming are summarised as follows:

1) Management and unions had quickly realised that the model was **not always realistic**, in that it could not model **soft constraints** ('nice to haves'). Also, it could not deal with the **social aspects** of rostering.

2) As users began to understand what was possible the constraints had become ever more **complex.** e.g. The model could not recommend shift start and finish times; it did not handle meal breaks well.
3) The LP could produce **only one solution**. Negotiations are obviously not very worthwhile if only one option is available. The unions were particularly unhappy about this aspect. This unhappiness was unnecessary, since there are usually many feasible roster patterns, for a given set of workload cover requirements.
4) Increasingly, the management wanted to apply the model more widely to more work areas. New applications required **re-modelling** of **LP formulation** that **only OR analysts could do**. It was also necessary for OR analysts to run the model to manage the integer heuristic when it did not work.
5) The users wanted to evaluate **more options**. They started saying "what if?", "yes but!". By necessity, the model became surrounded with **simple heuristic and even deterministic models**. These often included graphs to help explain what was going on in the model.

It goes without saying that these facts enforced some viewpoints shared by a number of OR professionals as well.

OR practitioners may take one final lesson from this story in British Airways. We believe that the disadvantages of LPs are also those of optimisation algorithms. What I leant from the interview definitely convinced me that my research direction was basically right. It is then necessary to study major computer based approaches (optimisation algorithms, heuristics, and simulation) to numerical optimisation problems, and their suitability in managerial decision making.

1.5 The objective of this research

A number of OR specialists are proud of their ability in problem-solving, especially for well-known standard problems. Some of them restrict themselves to standard problems so

as to gain recognition from colleagues within their academic circle. The crisis in OR has been forecast by several leading OR/MS professionals. Most of the work in Operational Research has been on the existence and finding of an optimal solution theoretically, while there should have been increasing emphasis on finding the best solution practically. Precisely the problem is partly caused by too much effort being wrongly put on pure optimisation.

Optimisation methods are not the only approach in managerial decision making. Two other closely-related approaches, central to numerical optimisation problems, are the heuristic and computer simulation methods.

Heuristic methods often determine good solutions to numerical optimisation problems. These methods play an important role in many managerial decision making problems.

Simulation is defined as a solution procedure that determines the best solution of a simulated problem. It may not be regarded as an efficient tool for optimisation problems, but becomes more important in managerial decision making with the development of information technology (IT).

Criteria commonly used to choose algorithms include accuracy, effectiveness, simplicity, and how economical they are. Referring to any algorithm, however, managers more often take into account efficiency, competency, and handiness. It can be seen that rules of thumb are used to select these three approaches. With conventional optimisation techniques, controversial heuristics, and formidable computer simulation, the question that has been raised during the last decade is:

"Which technique is more suitable for managerial decision making?"

Optimisation algorithms, heuristics, and computer simulation are considered as the available mathematical and computational tools. We will concentrate on why and how one should use them in certain circumstances.

In this research, we intend to identify feasible directions of promising areas for future research and application, and especially to question the position of optimisation algorithms in managerial decision making.

1.6 The layout of this research

A fairly general managerial problem can be formulated as a linear programming or non-linear programming model. Without the assistance of a computer, these optimisation algorithms can hardly be applied to a practical problem. Currently, there is a large amount of mathematical programming based software available. In Chapter 2 we intend to investigate the algorithms and their software associated with mathematical programming.

In order to compare blossoming heuristics, we will focus on investigating different heuristic thoughts in Chapter 3. A fully parallel study on heuristics is not possible since there are few commercial packages available although a large number of packages are coming forth at company or research level.

Simulations are even more difficult to analyse. Therefore, we will study the three procedures in computer simulation in Chapter 4.

In order to clarify major principles in the three approaches, we will undertake further examination through a case study. A number of heuristic methods are presented in order to illustrate some of the ideas discussed in Chapter 5.

Referring to methodology, modelling, feasibility, and applicability, we extensively discuss their roles in the three approaches to managerial decision making in Chapter 6.

2. Optimisation approach

2.1 Introduction

According to mathematical structure, we will discuss mathematical programming, which has been widely used in managerial decision making. Mathematical programming means different things to different people. To those interested in convex analysis, it is a branch of pure mathematics. To those interested in algorithms, it is a branch of numerical analysis. To those interested in the implementation of algorithms, it is a branch of computer science. To those interested in advising management, it is a branch of operational research.

Although mathematical programming does not solve all the world's problems, it does provide a convenient way to derive the quantitative conclusions that follow from a set of assumptions. So its application will continue to expand with the increasing availability of powerful computing facilities.

Many management applications are established linear programming (LP) models that are run regularly every quarter, month, week or even day as an established routine of management control. We noticed that LP codes can not handle many decision making problems because some constraints may not be so rigid as linear, the objective itself may be non-linear, and some variables may be uncertain. Many managers may regard non-linear programming (NLP) as a technical matter of no direct relevance to them. There is still lack of evidence that major progress has been made in non-linear optimisation. We notice that the progress over the last 30 years owes as much to better mathematics as to better computers.

We do not intend to discuss here dynamic programming in breadth, but note it often becomes a large-scale optimisation problem or can be handled by heuristics. Such a formulation containing an unusually large number of nonzero coefficients may cause

unexpected difficulties. We are interested in the main features underlying recent progress. These are triangular factorisation of the basis and the use of element pools. Thus we will discuss an important large-scale optimisation technique - decomposition, which can be applied to dynamic programming.

2.2 Mathematical programming approach

2.2.1 Linear programming and its software

a) Introduction

There are many applications of linear programming in managerial decision making. The basic problem of linear programming is to minimise a linear objective function of continuous real variables, subject to linear constraints.

For purposes of describing and analysing algorithms, the problem is often stated in the standard form

$$\min\{c'x : Ax = b, x \geq 0\},$$

where $x \in \Re^n$ is the unknown vector, $c \in \Re^n$ is the cost vector, and $A \in \Re^{m \times n}$ is the constraint matrix.

The feasible region described by the constraints $\{x : Ax = b, x \geq 0\}$ is a polytope, or simplex. It can be proved that at least one member of the optimal solution set lies at a vertex (extreme point) of this polytope.

After its discovery by Dantzig in the 1940s, the simplex method was unrivalled until the late 1980s for its utility in solving practical linear programming problems (Moore and Wright, 1993). Although never observed in practical problems, the worst-case behaviour of the algorithm is that the number of iterations or operations may be exponential in the

number of unknown variables. This poor performance led to an ongoing search for algorithms with better computational complexity.

This search continued until the late 1970s, when the first polynomial-time algorithm - Khachiyan's ellipsoid method appeared (Moore and Wright, 1993). Most interior-point methods, which we describe later, also have polynomial complexity.

b) The simplex method

This method generates a sequence of feasible iterates by repeatedly moving from one vertex of the feasible set to an adjacent vertex with a lower value of the objective function. The optimum can then be finally obtained at a vertex.

Beale (1984) noticed that "In nearly all practical mathematical programming problems, a typical variable occurs in not more than about 6 constraints. This is true whether the constraints are linear or non-linear, and whether the variables are continuous or discrete. So large problems are nearly always very sparse."

Computer programs for solving linear programming problems by the simplex method have existed since the early 1950s. They retain their central place in mathematical programming systems because successive implementations have exploited sparseness more and more efficiently. The steady progress that continues to be made in this process is remarkable.

The CPLEX, C-WHIZ, FortLP, LAMPS, LINDO, MINOS, OSL, and PC-PROG packages can be used to solve large-scale problems. Each of these packages accepts input in the industry-standard MPS format. Additionally, some have their own customised input format(for example, CPLEX LP format for CPLEX, direct screen input for PC-PROG). Others can be operated in conjunction with modelling languages (CPLEX, LAMPS, MINOS, and OSL interface with GAMS; LINDO and OSL interface with AMPL).

Recently, interfaces between spreadsheet programs and linear programming packages have become available. The What's Best! package links a wide range of standard spreadsheets (including Lotus 1-2-3 and Quattro-pro) to LINDO.

The IMSL and NAG libraries contain simplex-based subroutines. The BQPD package is aimed primarily at quadratic programming problems, but it does solve linear programming problems as a special case. It can take advantage of sparsity; as with the libraries above, but it is the user's responsibility to supply the problem data through subroutine arguments.

The packages LSSOL and QPOPT are aimed at linear least squares or quadratic programming problems but, as part of their capability, they can solve small to medium scale linear programming problems.

c) Interior-point methods

The announcement by Karmarkar (1984) that he had developed a fast algorithm that generated iterates that lie in the interior of the feasible set (rather than on the boundary, such as simplex methods). Since then, there has been intense research into a variety of methods that maintain strict feasibility of all iterates, at least with respect to the inequality constraints.

Interior-point products such as OB1, OSL, and KORBX have emerged and have proven to be competitive with, and often superior to, the best simplex packages, especially on large problems.

Some general patterns emerge in glancing at the survey of LP software. The most noticeable change is in the size of LP problems that can be solved on a PC. Compared to the earliest LP packages available for microcomputers, there has been an enormous

increase in the size of LP problems that can be tackled by a PC program. This has come about due to the extended memory addressing capabilities of software.

Most programs do not specify any internal limits on problem size any more. By adding memory, one can solve fairly large problems. According to one test mentioned by Sharda (1992), operational researchers were able to solve LP problems with as many as 367,000 non-zeroes; 16,000 rows or 69,000 columns on a desktop machine in about two hours. The practical time limit may be a more significant restriction than the advertised size limit in considering any of these programs. Software optimised for workstations solved the problem in even less time. This suggests that serious optimisation is rapidly becoming practical on a desktop computer.

One other limitation is in the ability to move large problems from one computer to another using secondary storage media. This proved to be a bigger bottleneck than the capacity of the software. Of course, one can use the super high capacity disk drives or portable hard disks to move problems from one computer to another. Until these technologies become affordable and widely available, networks are obviously the only way to move problems from one environment to another. Thus the problem size capacity does not appear to be a significant issue any more.

Sharda (1992) surveyed linear programming software for personal computers. The availability and capability of linear programming (LP) software has kept pace with the growth in computer hardware technology. In order to keep OR/MS professionals abreast of recent development in this area, OR/MS Today initiated a new service in October 1990 by publishing a survey of LP software for desktop computers. This is an update of the earlier survey. It indicates that several exciting trends in LP software are emerging.

Referring to hardware requirements, if a program can run on any of the IBM-PC compatible computers (XT/AT and higher), then it is able to run on PC/MS-DOS

machines. If a program requires a computer based on an Intel 80386 or higher processor, it is regarded as another class. Obviously, all programs capable of running on "IBM compatible machines" can run on 80386/80486 machines. Several programs such as Best Answer^{TM}, GINO, LINDO, LINGO, Microsoft Excel 4.0, What's Best! and XA, are available for the Macintosh environment and workstations.

2.2.2 Non-linear programming and its software

a) Introduction

Considerable progress has been made recently on hill-climbing methods for finding possibly local optima for non-linear functions, subject to linear or non-linear constraints.

Either Quasi-Newton or conjugate gradient methods can be used to solve unconstrained optimisation problems, assuming that function values and first derivatives, but not second derivatives,can conveniently be computed.

A consensus seems to be emerging about the general strategy for adapting these methods to constrained optimisation. The trial solutions should satisfy linear constraints. They should also satisfy linear approximations to non-linear constraints, with the non-linearities thrown into the objective function through the Lagrangean multipliers.

We need not be concerned with these methods if our non-linear optimisation problems are all small, on the grounds that a primitive algorithm should be able to solve a small problem on a powerful computer. But this is a dangerous conclusion, since inefficient algorithms usually produce very inaccurate answers because of round-off errors.

b) Unconstrained optimisation methods

The unconstrained optimisation problem is central to the development of optimisation software since constrained optimisation algorithms are often extensions of unconstrained algorithms. In the unconstrained optimisation problem

$$\min\{f(x) : x \in \Re^n\}$$

we seek a local minimum of a real-valued function f defined on $\Re^n$, that is, a vector $x \in \Re^n$ such that $f(x^*) \leq f(x)$ for all $x \in \Re^n$ near x^*. We do not discuss global minimisation algorithms because at present there is no widely available code for global minimisation.

The well-known algorithm is probably the Newton method based upon the computation of the gradient vector and/or the Hessian matrix. According to Moore and Wright (1993), versions of Newton's method are implemented in BTN, GAUSS, IMSL, LANCELOT, NAG, OPTIMA, PORT 3, PROC NLP, TENMIN, TN, TNPACK, UNCMIN, and VE08. These codes enforce convergence when the initial point is not close to a minimise by using either a *line-search* or a *trust-region* approach. These two approaches differ mainly in the way they treat indefiniteness in the Hessian matrix.

Nevertheless, Quasi-Newton methods can be used when the Hessian matrix is difficult or time-consuming to evaluate. Instead of obtaining an estimate of the Hessian matrix at a single point , these methods gradually build up an approximate Hessian matrix by using gradient information from some or all of the previous iterates visited by the algorithm. GAUSS, IMSL, MATLAB, NAG, OPTIMA, and PROC NLP implement Quasi-Newton methods.

c) Constrained optimisation methods

The general constrained optimisation problem is to minimise a non-linear function subject to non-linear constraints.

$$\min\{f(x) : g_i(x) \leq 0,\ i \in I,\ g_i(x) = 0,\ i \in E\},$$

where each $g_i(x)$ is a mapping from $\Re^n$ to $\Re$, and I and E are index sets for inequality and equality constraints respectively.

The main techniques that have been proposed for solving constrained optimisation problems are *reduced-gradient* methods, *sequential linear* and *quadratic programming* methods, and methods based on *augmented Lagrangeans* and exact penalty functions.

In order to express first-order and second-order conditions for a local minimum, the Lagrangean function is defined as

$$L(x,\lambda) = f(x) + \sum_{i \in I \cup E} \lambda_i g_i(x).$$

The well-known first-order necessary conditions for the existence of a local minimum x^* of the constrained optimisation problem require the existence of Lagrangean multipliers λ_i^* such that

$$\nabla x L(x^*,\lambda^*) = \nabla f(x^*) + \sum_{i \in A} \lambda_i^* \nabla g_i(x^*) = 0,$$

where

$$A = \{i : i \in I, g_i(x^*) = 0\} \cup E,$$

$$\lambda_i^* \geq 0, \qquad i \in A \cap I.$$

The second-order sufficiency condition requires that (x^*,λ^*) satisfy the first-order condition $\nabla x L(x^*,\lambda^*) = 0$ and that the Hessian of the Lagrangean

$$\nabla^2 L(x^*,\lambda^*) = \nabla^2 f(x^*) + \sum_{i \in A} \lambda_i^* \nabla^2 g_i(x^*),$$

satisfy $w' \nabla^2 L(x^*,\lambda^*) w > 0$ for all non-zero w in the set

$$\{w: \nabla g_i(x^*)' w = 0, i \in I^+ \cup E; \nabla g_i(x^*)' w \leq 0, i \in I^0\},$$

where

$$I^+ = \{i \in A \cap I : \lambda_i > 0\},$$

$$I^0 = \{i \in A \cap I : \lambda_i = 0\}.$$

Optimisation packages are based on different optimisation algorithms. Here we can only name some of them. The sequential quadratic programming algorithm is a generalisation of Newton's method for unconstrained optimisation in that it finds a step away from the current point by minimising a quadratic model of the problem. A number of packages, including NPSOL, NLPQL, OPSYC, OPTIMA, MATLAB, and SQP, are found with this approach. The OPTIMA and OPTPACK libraries also contain augmented Lagrangean codes. Reduced-gradient algorithms avoid the use of penalty parameters by searching along curves that stay near the feasible set. The standard reduced-gradient algorithm, implemented in CONOPT, searches along the steepest-descent direction in the super basic

variables. The generalised reduced-gradient codes GRG2 and LSGRG2 use more sophisticated approaches.

Optimisation software is progressing at a rapid rate. Growing availability of linear and non-linear programming techniques in spreadsheets is an encouraging indicator of acceptance of OR/MS approaches. Other developments in modelling languages, user interfaces and parallel processing suggest that some modelling efforts are being facilitated through optimisation software.

As Beale (1984) said, a typical variable occurs in not more than about 6 constraints in nearly all practical mathematical programming problems. Thus large problems are nearly always very sparse. General mathematical programming systems must be applicable to large problems, so they must use algorithms that exploit sparseness efficiently, even if this makes them slower than special-purpose programs on small dense problems.

2.2.3 Decomposition techniques for large-scale optimisation problems

a) Background

Decomposition was recognised as a natural tool for handling large-scale linear programming problems. If a special structure can be identified, decomposition can often be used to reduce a large-scale problem to components of a more manageable size, or to admit enhancement of solution procedures. This review provides a historical perspective as well as recent LP decomposition approaches, including its classification and some implementations.

The idea of decomposition was first proposed by Dantzig in 1959, and subsequently many research results have been published. The decomposition principles of Dantzig-Wolfe (1960) lead to algorithms that transform the original problem into a sequence of subproblems corresponding to the uncoupled subsystems. The primal or Benders

decomposition method (1962) was also introduced shortly after the Dantzig-Wolfe decomposition method was developed.

Some computer-based algorithms/programs for decomposition are reported, research into decomposition of linear programs using parallel computation has also been reported by Ho, Lee and Sundarraj (1988).

In contrast to the linear case, the application of nested primal or dual decomposition to non-linear programs and stochastic problems has received comparatively little attention. Many LP based decomposition techniques have been proposed for different purposes. Basically, we classify decomposition into Dantzig-Wolfe (dual) and Benders (primal) decomposition, cross decomposition, LU decomposition, and Bartels-Golub decomposition.

b) Dantzig-Wolfe (dual) and Benders (primal) Decomposition

Dantzig and Wolfe (1960) developed a decomposition principle that imposes on each subproblem additional constraints so that the optimal solution of one subproblem is independent of the solutions of the other subproblems. The additional constraints are selected so that the union of the solutions of the subproblems will be optimal for large-scale problems.

It is well known that Dantzig-Wolfe decomposition and Benders decomposition are dual pairs, i.e., the Benders algorithm (1962) that is applied to pure linear programming, coincides with Dantzig-Wolfe decomposition algorithm applied to the dual of this problem. The decomposition principle in its primal or dual form has provided very efficient algorithms for many MIP problems. The technique allows advantage to be taken of the special structure of the problem by solving a sequence of subproblems. Hence, one exploits either the primal or the dual substructure of the problem.

The subproblems are co-ordinated by a master problem corresponding to the global constraints through primal (proposal) and dual (prices) information. While the application of primal decomposition on the dual problem is equivalent, in terms of final results, to the use of dual decomposition on the primal, the two methods can differ significantly from a computational point of view. In particular, models with a large number of columns and a comparatively small number of rows will require a smaller basis when handled through the primal method. In contrast, column generation is more compatible with the matrix representation adopted in commercial codes, than the addition of cuts. This would favour the use of the dual problem.

Another dual decomposition is known as the Lagrangean relaxation method. Geoffrion (1974) showed that the method can be generalised via Lagrangean relaxation to deal with mixed integer programming. The dual subproblem is obtained by taking the Lagrangean relaxation of the original problem relative to some constraints. Each iteration consists of 1) selecting a new set of Lagrangean multipliers by the dual master problem and 2) solving the dual subproblem for given values of the multipliers. The dual decomposition algorithm solves successively a "dual subproblem" and a "master problem" until the optimum is achieved and verified. This algorithm may become computationally attractive if the "complicating" constraints are relaxed so as to obtain a relatively easy-to-solve sub-problem. The dual decomposition algorithm effectively solves the formal Lagrangean dual relative to the given subset of constraints. Hence , when applied to MIP, a duality gap may arise and customarily is closed by an enumeration scheme of the branch-and-bound type. Lower bounds on the value of the (minimisation) problem are obtained at every iteration of the dual decomposition algorithm instead of solving the formal Lagrangean dual optimally.

c) Cross Decomposition

The complete cross decomposition algorithm is developed for solving mixed integer linear programming problems. Van Roy (1983) proposed the basic idea underlying cross

decomposition to use both subproblems in one single decomposition procedure as follows: (PS) and (DS) are the primal and dual subproblems respectively.

(1) Initialise. Select initial values for the Lagrangean multipliers and set up the corresponding (DS).

(2) Solve the (DS). Perform convergence test CT_P: either stop, or go to 4), or set up the (PS) corresponding to the optimal solution of the current (DS) and go to (3).

(3) Solve the (PS). Perform convergence test CT_D: either stop, or go to (4), or set up the (DS) corresponding to the optimal dual solution of current (PS) and go to (2).

(4) Master problem. Find new values for either the Lagrangean multipliers, or the primal variables that are held fixed in (DS) or (PS). Set up the corresponding subproblem and go to (2) or (3) respectively.

Actually, the convergence tests CT_P and CT_D control the sequence of primal and dual subproblems as well as the master problem.

Not every algorithm that employs primal and dual subproblems relates to cross decomposition, e.g., the branch and bound procedure, primal-dual ascent algorithm by Fisher, Northup, and Shapiro (1975).

Holmberg (1992) developed a modification of cross decomposition, called mean value cross decomposition for linear programming problems. The method is a generalisation of the Kornai-Liptak (1965) method and eliminates the need for using master problems. The base for the method is the subproblem phase in cross decomposition, where performance between the dual subproblem and the primal subproblem is iterated. However, instead of using the last solution of one subproblem as input to the other and vice versa, the average (mean value) of all previously obtained solutions is used. It is shown that this is equivalent to the Brown (1949) and Robinson (1951) method for a matrix game, and this fact is used to prove convergence of the procedure.

d) LU Decomposition

LP models may represent large, complex systems consisting of independent subsystems coupled by global constraints. Such LP problems are said to have a block-angular structure. We first consider solving a large set of simultaneous linear equations

$$Ax = b$$

where A is a sparse matrix of order n. If the matrix A can be factored as a product of lower and upper triangular matrices, that is

$$A = LU,$$

then a solution for x can be obtained by forward and backward substitution of the two triangular systems:

$$Ly = b,$$

$$Ux = y.$$

This is the so called LU decomposition.

In large-scale linear programming the computation time is greatly influenced by the method of finding an accurate and compact form of the inverse of a sparse matrix B. It has been recognised that if the product form of B is used, and in particular the LU form, then, by pivoting first on the diagonal elements of U in reverse order, a product form representation can be obtained that does not reduce the degree of sparseness.

A more extensive decomposition to solve the above linear equations is accomplished by decomposing the matrix A into a product

$$A = PLUQ$$

where P and Q are permutation matrices, L is a lower-triangular matrix with units on its main diagonal, and U is an upper triangular matrix. According to this decomposition, the system may be written in equivalent form:

$$Px(1) = b,$$

$$Lx(2) = x(1),$$

$$Ux(3) = x(2),$$

$Qx = x(3)$,

Each subsystem can be easy solved by eliminating the subsidiary vectors x(1), x(2), and x(3).

LU decomposition is effective compared to various types of preliminary transformations to eventually obtain serial relationships among the original equations in terms of the x's only.

e) Bartels-Golub Decomposition

Bartels (1971) first considered a stabilisation of the simplex method for handling linear programming bases in the sparse case, i.e., many zeros in those bases. Probably the best-known research is that of Forrest and Tomlin (1977) on updating triangular factors of the basis to maintain sparsity in the product form simplex method. Goldfarb (1977) studied some properties on the Bartels-Golub decomposition for linear programming bases.

Reid (1982) describes a sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases. It includes interchanges that, whenever this is possible, avoid the use of any elimination (with consequent fill-ins) when revising the factorisation at an iteration. Test results on some medium scale problems are presented and comparisons made with the algorithm of Forrest and Tomlin. Reid's algorithm has "better stability and fill-in properties than the closely related algorithm of Forrest and Tomlin" although numerical experiments indicate that these advantages are usually quite slight in practice.

f) Some implementations based on decomposition

The decomposition technique was proposed for travelling salesman problems, scheduling, multi-commodity networks, facility location, lot sizing, set partitioning, and matching etc.

Ho and Loute (1983) described their computational experience with codes DECOMPSX and LIFT which are built on IBM's MPSX/370LP software for large-scale structured programs. DECOMPSX is an implementation of the Dantzig-Wolfe decomposition algorithm for block-angular LP's. LIFT is an implementation of a nested decomposition algorithm for staircase and block-triangular LP's. A diverse collection of test problems drawn from real applications is used to test these codes, including multinational energy models and global economic models.

Ho, Lee and Sundarrj (1988) reported their DECOMPAR code: an implementation of the Dantzig-Wolfe decomposition algorithm for block-angular linear programs using parallel processing of the subproblems. The software is based on a robust experimental code for LP decomposition and runs on the CRYSTAL multicomputer at University of Wisconsin-Madison. Their initial computational experience is also reported.

Cross decomposition was first implemented by Van Roy (1982) for solving capacitated facility location problems. The algorithm performed well on a common set of standard test problems, compared with alternative algorithms.

Kaliski and Ye (1993) developed a short-cut potential reduction algorithm for linear programming. They examined decomposition techniques which greatly reduce the amount of work required by such interior point methods as the dual affine scaling and the dual potential reduction algorithms. In an effort to judge the practical viability of the decompositioning, the performance of the dual potential reduction algorithm with and without decompositioning is compared over a set of randomly generated transportation problems. Accompanying a theoretical justification of these techniques, the implementation details and computational results of one such technique are presented.

By using decomposition, the subproblems in a large-scale LP problem have the property that any key matrix can be triangularized. This allows network-like techniques to be

implemented on nodes of existing multicomputers to handle the subproblems. Using a multicomputer with multiple nodes, the decomposition approach is expected to be viable.

2.3 Summary

To operational research analyst interested in the implementation of algorithms, it is a combination of operational research, numerical analysis, and computer science. We discussed computer based mathematical programming in this Chapter. We focus on linear programming and its software arround the simplex method and interior-point methods. We study non-linear programming and its software through unconstrained optimisation methods and constrained optimisation methods. Instead of discuss dynamic programming directly, we discussed an important large-scale optimisation technique - decomposition, Dantzig-Wolfe (dual) and Benders (primal) Decomposition, Cross Decomposition, LU Decomposition, Batels-Golub decomposition, and some computer implementations based on decomposition, which are applied to large-scale optimisation problem. In the appendix, we attatched some further details of the software mentioned here.

3. Heuristic Methods

3.1 Introduction

3.1.1 Background

Recently, attention in the literature has been increasingly focused on so-called heuristic methods as an aid to decision-making. As the following passage demonstrates, the literature contains sharply diverging views about heuristic methods in theory and in practice. It should be pointed out that there is no clear, objective way of establishing whether or not a particular definition is correct.

It is said as early as 300 A.D. Pappas wrote to Euclid, and suggested an approach of approximate methods - which are easy to use but which do not guarantee optimality. The subject - heuristics has gradually been accepted through the work of well-known mathematicians Descartes and Leibniz.

3.1.2 Clarification of the concept

There are many definitions of a heuristic method of problem solution. Researchers in different fields often define heuristics in different ways. The following selection of definitions should illustrate how inconsistently the term "heuristic methods" is used in the literature.

Nicholson (1971) defines a heuristic method as a procedure "... for solving problems by an intuitive approach in which the structure of the problem can be interpreted and exploited intelligently to obtain a reasonable solution."

Tonge (1961) uses a definition: "by heuristics we mean principles or devices that contribute, on the average, to reduction of search in program-solving activity. Heuristic

programming is the construction of problem-solving programs organised around such principles or devices."

Wiest (1966) formulates his definition as follows: "in simplest terms a heuristic program is a collection or combination of heuristics used for solving a particular problem", whereby "we may describe a heuristic as any device or procedure used to reduce problem-solving effort - in short, a rule of thumb used to solve a particular problem"

Minsky (1967) proposes that: "A program which independently decides what should be done next is referred to as 'heuristic'."

Newell (1969) defines heuristic methods as "programs that performed tasks requiring intelligence when performed by human beings."

Wheeling (1969) ascribes the attribute "heuristic to any procedure (which) may sometimes fail."

According to McMillan (1970), a heuristic method is "a computational procedure which, when applied to any one of the class of problems for which it is applicable, will yield a good solution in a finite number of steps."

Hinkle and Kuehn (1970) describe a method as heuristic "which searches for a satisfactory rather than an optimal solution."

With reference to Klein (1971), Kirsch formulates his definition as follows: "A heuristic program is a usefully defined problem-solving method with great heuristic power and without any guarantee of finding a solution" (Kirsch,1971, pp.157).

Müller-Merbach (1971, 1973a, 1973b) describes heuristic methods simply as methods of approximation. However, if the two terms refer to the same methods and can be extended to refer to the same methods, and are thus identical, it would be better to use the old, well-known term "methods of approximation". If they are, however, not identical, it is not clear whether heuristic methods are merely a subset of the methods of approximation, or whether the methods of approximation represent only some of the possible heuristic methods.

Herroelen (1972) uses the following definition: "since then the terms 'heuristic' and 'heuristic method' are used to describe each rule of thumb, strategy, trick, simplification or any other means, that may reduce the effort in the search for solutions of complex problems by the elimination of possible but less interesting solution alternatives and thus may lead to useful solutions that are usually non-optimal."

Beier (1973) explains the heuristic method thus: "Heuristic decision-making methods are systematic, - i.e. non-mathematical problem-solving procedures which attempt to solve a particular class (domain) of problems or very specific problems with the help of general or specific heuristic rules (principles, strategies or procedural instructions), but which are unable to guarantee an acceptable solution in individual cases and which can never guarantee an optimal solution. A heuristic rule is a systematic procedural instruction which tries, among other things, to reduce the complexity of the problem being solved and to structure the procedure. Its objective is to arrive at an acceptable solution to the problem within a pre-set time period, taking into account known methods of decision-making and existing processing capacity."

Silver, Vidal, and Werra (1980) once defined a heuristic method as "a procedure for solving a well-defined mathematical problem by an intuitive approach in which the structure of the problem can be interpreted and exploited intelligently to obtain a reasonable solution."

Foulds (1983) proposed ".....heuristics means a method which, on the basis of experience or judgement, seems likely to yield a good solution to a problem but which cannot be guaranteed to produce an optimum."

Foulds (1983) suggested 5 possible ways to the strategy of developing and using approximate methods to solve a complicated NP problem:

(1) find an efficient algorithm for the problem;

(2) show that only special cases of the problem are of interest and find an efficient algorithm for them;

(3) relax some of the constraints of the problem and develop an algorithm for the relaxed (easier) problem;

(4) construct an algorithm that runs quickly on most of the problems likely to be encountered;

(5) give up the quest for optimality and provide a method which runs quickly and produces useful but not necessarily optimal solutions.

It is extremely unlikely that aim (1) can then be achieved. Aims (2) and (3) are usually more appropriate. However, Churchman (1970) has quite rightly warned of the dangers of discarding the problem at hand for a surrogate which is not equivalent but which can be solved by known methods. This practice has raised the criticism that sometimes O.R. practitioners "bend a client's problem" so that it can be solved by a standard method, thereby producing solutions of little interest to the client. Aim (4) is often attempted by O.R. consultants and often achieves customer satisfaction. The simplex algorithm of linear programming is a good illustration. Unfortunately, sometimes such an algorithm cannot be found. He then discussed the condition of using a heuristic approach - what can be done when we are reduced to aim (5).

Ballou (1989) also found that heuristics are not difficult to specify because researchers are in the habit of using them constantly in daily activities.

This list of definitions is sufficient to illustrate that there is disagreement in the literature about which characteristics of a heuristic method uniquely define it as "heuristic". It is therefore also unclear how the term can be extended to cover other methods which claim to be "heuristic".

According to Newell, Tonge, and Minsky, the calculations involved in differential calculus or in the simplex method would be heuristic methods, because they without doubt reduce in an "intelligent way" the search effort required to find a solution.

Wheeling's view is also problematic, because it does not explain whether methods should also still be described as heuristic if they produce no solution as a result of being prematurely interrupted, e.g. because of lack of time or money.

Hinkle/Kuehn and McMillan emphasise the search for good or satisfactory solutions. A solution is satisfactory when it meets the requirements of the individual decision-maker. However, because different decision-makers usually have requirements which differ in their stringency, these attempts to produce a definition cannot differentiate between "heuristic" and "non-heuristic" methods with any inter-subjective clarity.

Heuristic methods are, according to Wiest and Herroelen, those methods which contain "rules of thumb"; Klein uses instead the phrase "heuristic principles". Kirsch maintains that heuristic methods are characterised by a "great heuristic power", by which he means, that these methods can, in a particular length of time, solve a great number of the problems belonging to the class of problems for which they were developed. Since, however, this formulation offers no quantitative information about the number of problems to be solved or about the length of time to be specified, it also lacks inter-subjective clarity. These comments are also valid for Beier's very detailed definition. Furthermore, his remarks about the essential characteristics of heuristic rules are not very

instructive, because most of the methods that are universally regarded as "non-heuristic" share the sort of rules that Beier claims are particularly characteristic of heuristic methods.

It is thus clear from these examples that a definition of the concept is urgently required. One reason for the lack of precision in the definitions is that concepts are often used in the definition which themselves need to be defined. Another reason is that every one of these attempts at a definition lacks the necessary refinement to unambiguously distinguish heuristic from "non-heuristic" methods.

Note that the some of above definitions are out of our discussion. It is not an easy task to examine what the vaguely used term "heuristic methods" means in concrete terms within a theoretical framework. We do not intend to ally heuristic with logic, philosophy, or psychology, where it aims at investigating invention and discovery. *A heuristic is therefore defined as an algorithm which it cannot be mathematically proved to find an optimal solution for a problem.*

3.1.3 Classification of heuristics

People use similarity, precision, fruitfulness and simplicity as criteria for the heuristic approach. However, what is the essence of heuristics? In this research we propose that substitution is the essence of any heuristics. Heuristics must be often simpler than any known optimisation algorithms (if exist). This simplification is carried out on

(1) prototype - problem substitution, or

(2) mathematical model - model substitution, or

(3) known or unknown algorithm - algorithm substitution.

Based on this idea, we attempt to classify major heuristics and clarify some implications of these heuristics in this chapter.

3.2 Heuristic approach

3.2.1 Problem substitution heuristics

On the basis of experience or judgement, if a numerical optimisation problem is difficult to solve, we can substitute the original problem with a solvable problem. So-called *knowledge-based heuristics* and *common sense heuristics* may fall within this type of heuristics, which constitutes the most interesting area in managerial problems.

Generally, knowledge representation methods may be practised as three tasks:

1) declarative knowledge, for rule based reasoning,
2) heuristic methodologies, for the deterministic and context-dependent decision making, and
3) strictly algorithmic procedures, for the tasks inclined to data abstraction and mathematical manipulations.

Knowledge-based heuristics are based upon the understanding of the background of the optimisation problem.

Distribution and scheduling problems with dynamic or combinatorial structure often fall in NP-hard. In order to solve these problems, expertise or common-sense is often a source of inspiration.

Good heuristics are frequently the result of common sense procedures that work effectively. They may be based on concepts, principles, and theories that relate to a specific problem, or they may result from observing the form of optimised solutions and mimicking them. It is not easy to recognise when good heuristics have been found, since testing them against other methods is difficult when practical size problems are involved.

3.2.2 Model substitution heuristics

We consider so-called *model relaxation with bounding method*, as one of model substitution heuristics, which relaxes the mathematical model of the optimisation problem when the optimum solution can not be found. Hence, it has to provide a bound on the value of the optimal solution, i.e., the value of the optimum solution can not be better than the bound.

We also put *model relaxation without bounding method* into this category, which relaxes the original model out of some simple rules. For example, Silver, Vidal, and Werra (1972) presented two illustrations: 1) In an integer linear programming problem, a bound can be obtained by ignoring the integer constraints and solving the much simpler, continuous variable problem. 2) In a travelling salesman problem the difficulty of solution is caused by the constraint that every city must be visited precisely once in a single tour. Removing the single tour constraint leads directly to a bound on the original problem.

Whenever the original problem is substituted its model will be naturally different. Model relaxation with bound implies a restricted relaxation, i.e., even if a relaxed solution itself may not be found, its objective value, or a bound must be determined. The value of an optimal solution must lie between the value of the heuristic solution and the bound. However, it is wrong to suppose that if the value of the heuristic solution is very close to the bound, it must be very close to an optimal solution. Actually, a heuristic solution itself constitutes an opposite bound for optimal solution.

Another type of heuristic is to relax the mathematical properties of the original model without bounding. Here, In Table 3.2.2a, we summarised the possible ways to substitute the original characteristics of the mathematical model.

Original characteristics	Substituted characteristics
NP	P
Discontinuous	Continuous
Non-smooth	Smooth
Integer	Mixed
Discrete	Non-discrete
Random	Determinative
Infinite	Finite
Capacitated	Incapacitated
Multiple	Single
Derivable	Differentiable

Table 3.2.2a Substitution Pattern

3.2.3 Algorithm substitution heuristics

This kind of heuristics could be the *extreme value method* case, which generates many solutions and chooses the best. The so-called *improvement method* can also be included in this heuristic, which searches repeatedly for a better solution on the basis of a feasible solution. Also, it could be the *modified mathematical programming method*, which simplifies the whole optimisation procedure so as to save computational effort

Substitution is not merely approximation. The question now arises as to the precise nature of a heuristic problem-solving method. Some of the definitions cited in the previous section regard the reduction of the search effort required to find a solution as an essential characteristic of a heuristic method.

What is clearly meant by this is that a heuristic method does not generate every possible node on the problem-solving tree. In an extreme case, its application may result in an

straight, unbranched solution path. If one describes the terminal nodes of the general problem-solving tree as candidate solutions, the following statement is also valid: heuristic methods are structured in such a way, that some of the candidate solutions are not generated. It is thus established that the enumeration of all possible solutions is not a heuristic method. The only criterion that has so far been established - that of neglecting candidate solutions in the search process - is not sufficiently precise to allow any further statements to be made about heuristic methods. However, since there are many other methods apart from that of exhaustive enumeration which are not classified in the literature as heuristic, the definition of a heuristic method must contain further defining characteristics. One such characteristic referred to in the literature is the lack of a guaranteed optimal solution. In other words, a feature which distinguishes heuristic methods from other methods is that applying them does not guarantee that an optimal solution will be arrived at. Therefore, of all the known methods, all those can be eliminated as non-heuristic which converge on an optimal solution in a *finite* number of transformations.

Among these are the classical analytical methods and the numerical-iterative linear optimisation methods, to name but a few examples. In addition to these, there are a number of methods for solving particular types of problem which usually only converge after an *infinite* number of steps (e.g. methods for solving uni-dimensional problems: *Fibonacci* method, Golden Section method, and most gradient methods). Since in practice every search must be interrupted after a finite number of transformations, one cannot guarantee that this group of methods will produce a solution either. Nevertheless, the author's view is that these methods should not be described as "heuristic", because they allow an approximation to the required optimal solution to be calculated to any degree of accuracy. In other words: after the procedure has been interrupted, a correspondingly small neighbourhood can be identified, within which the solution must lie. According to the view represented here, these methods are therefore not heuristic methods, but rather approximation methods. Equally, one must conclude from this that

heuristic methods are not approximation methods, because they offer no proof of convergence.

Some heuristic methods are designed for solving a special class of decision-making problems, namely, problems which involve extreme values and upon which supplementary conditions are imposed. These are characterised by a further property, and this will become clear by comparing them with the corresponding convergent methods.

A distinction is normally made in relation to such problems between feasible and optimal solutions. A result of the problem-solving process is described as feasible when it complies with the constraints; any permissible result where the objective function has an extreme value is optimal. If one compares the heuristic methods designed to solve this type of problem with the corresponding non-heuristic methods which guarantee a solution, the following difference emerges: many convergent methods, e.g., the dual simplex and Gomory's sectional plane methods, only produce a feasible solution in the last transformation, a solution which is however also optimal. Therefore, if the search procedure is interrupted prematurely one cannot even get a feasible solution.

In contrast, most of the special heuristic methods for this type of problem-solving place particular emphasis on achieving and retaining *feasibility*. Because this property is not essential for our definition of heuristic methods, although numerous such methods possess it, it should also only be seen as peripheral.

We can see that problem-solving methods should only be described as heuristic when they

(1) work with the help of non-arbitrary decision variables;

(2) exclude potential solutions from the search process (leave some nodes unexplored);

(3) cannot give proof of convergence.

Thus, methods which actually do converge are also heuristic, as long as convergence cannot be proved. Methods which were originally heuristic, but which prove to be convergent, can therefore become finite methods with a guaranteed solution or approximation methods.

The use of such heuristic methods is motivated by the emergence of the theory of NP-completeness, i.e. the study of the complexity of algorithms. There is usually a finite number of feasible solutions for a combinatorial problem. For example, a logistics manager must, among other things, plan the structure of the logistics network, set inventory policies, select transportation modes, set plans for contingencies to assure that plans are being met.

3.3 Summary

In this Chapter we listed a number of definitions for heuristic, discussed the difference among them, and clarified of the concept. A principle - substitution is proposed so as to classify heuristics. Many well-known heuristics are therefore categorised as Problem substitution heuristics, Model substitution heuristics, and Algorithm substitution heuristics. Heuristic approach is then studied according to this classification.

4. Computer simulation

4.1 Introduction

4.1.1 Background

Computer simulation methods have been developed since the early 1960s and may well be the most commonly used of all the analytical tools of operational research. Computer simulation is a technique used to model the operation of a system and employs a computer program to model the operation and perform simulation computations.

Then why simulate when it will be time consuming and there may be alternative approaches? In many problems the mathematical manipulations required to derive consequences from a symbolic model can be carried only to a point where the optimising values of the variables are stated in terms of a complex functional relationship. As with other models that do not utilise an explicit mathematical calculus, optimal combinations of controllable variables must be found by a search process or some other form of enumeration. This is a consequence of the structure of the problem, for if there were a mathematical theory for finding the optimum, simulation would not be needed.

Though simulation can be time consuming and therefore expensive in terms of skilled manpower, real experiments may also turn out to be expensive, particularly if something goes wrong! Admittedly it takes a significant amount of time to produce working computer programs for simulation models. However, once these are written then an attractive opportunity presents itself. Namely, it is possible to simulate weeks, months or even years in seconds of computer time. Hence a whole range of policies may be properly compared.

Anderson et al (1991) gave some of the reasons why computer simulation is so widely used:

(1) It can be used for a large number of practical problems.

(2) It can obtain good solutions to problems that are too complex to be solved with procedures such as linear programming, waiting line models, or inventory models.

(3) The simulation approach is straightforward and hence is relatively easy to explain and understand. As a result management confidence is increased, and consequently, acceptance of the model is more easily obtained.

(4) Computer manufacturers have developed extensive software packages consisting of specialised simulation programming languages, thus facilitating use of simulation in practice.

Today management scientists are not easily separated from their computers and with good reason. Computers have become smaller, cheaper, more powerful and easier to use by non-specialists. In particular, the development of powerful and cheap portable machines has opened up wide areas of work for the management scientist. With other advantages, computer simulation may become an inevitable approach to numerical optimisation problems.

Computer simulation is applicable in complex cases where analytical procedures cannot be employed. The simulation model and simulator provide a convenient experimental laboratory. Simulations are often used to compare the performance of different potential solutions to a problem. We think that computer simulation, even without absolute optimisation, is important because it results in a forward-looking point of view; it tries to provide the tools that will permit the decision maker to arrive at better decisions in the future.

4.1.2 Clarification of the concept of computer simulation

Computer simulation is a trial-and-error procedure; a variety of values is generated for the decision variables, and the best of the feasible solutions is chosen. The principles of

computer simulation are simple enough. Management scientists build a model of the system of interest, write computer programs which embody the model and use a computer to imitate the system's behaviour when subject to a variety of operating policies. Thus the most desirable policy may be selected.

Computer simulation provides a tool for evaluation and comparison of alternative basis of action during decision making. It can be used as an approach to a numerical optimisation problem.

However, there are some different opinions on this issue. Anderson et al. (1991) concluded that "The computer simulation is one of the most frequently used management science tools but *computer simulation should not be viewed as an optimisation technique*."

Pidd (1988) pointed out that "Management scientists tend to employ mathematical and logical models rather than scale models. These represent the important factors of a system by a series of equations *which may sometimes be solved to produce an optimal solution.*" In fact, such a series of equations may sometimes to be solved to produce an optimal solution. This is true when, for example, the model fits the structure required for linear programming or non-linear programming.

Although computer simulation requires an additional expenditure for computing, the results can be shown to produce the optimum which mathematical theory predicts. If formulae for specific values cannot be derived, it is possible to obtain the optimum values of the variables by means of a numerical approximation. Computer simulation provides a tool for evaluation and comparison of alternative courses of action during decision making, i.e., which can be applied to a numerical optimisation problem.

The word optimal solution or optimum in computer simulation may be somewhat misleading. It does not mean the optimal solution in any absolute sense, but refers instead to the best solution that the decision maker can attain with the resources and time available. His method of optimising is often intuitive and therefore not explicit.

4.1.3 Categorising computer simulation

The real world is rarely kind enough to allow precise replication of an experiment. One of the skills employed by physical scientists is the design of experiments which are repeatable by other scientists. This is rarely possible in management science. It seems unlikely that an organisation's competitors will sit idly by as a whole variety of pricing policies are attempted in a bid to find the best. It is even less likely that a military adversary will allow a replay of a battle.

Simulations are precisely repeatable. Furthermore, one of the objectives of a simulation study may be to estimate the effect of extreme conditions and to do this in real life may be dangerous or even illegal. An airport authority may take some persuading to allow a doubling of the flights per day even if they do wish to know the capacity of the airport. Simulated aircraft cause little damage when they run out of fuel in a simulated sky.

First of all, we categorise computer simulation according to the major objectives:

1) Problem-solving;
2) Problem-finding;
3) Problem-understanding.

Models that are useful to decision makers must predict the consequences of specific actions or inputs. The ability to predict accurately a system's behaviour is not sufficient to satisfy the decision maker's needs. They always desire the ideal action - that combination of inputs which will most nearly obtain the goals. Throughout this research, we will focus on computer simulation for problem-solving although it is difficult to distinguish these purposes.

In problem-solving, the simulation work of such a project can be viewed as having three phases: modelling, programming, and experimentation. Precisely, logical models are usually required though in the case of system dynamics these are expressed in mathematical form, which behave like or simulate the real system; at this stage, great care is taken to ensure that the computer simulation model is descriptive of the real system. Accordingly, they have to make a simulation program that can be executed in a computer environment. Then, through a series of computer runs, or experiments, they learn about the behaviour of the simulation model. The characteristics that are observed in the model are then used to make references about the real system. Because simulation is an experimental approach, modelling and programming can be regarded as preliminaries to the real business of simulation.

4.2 Computer simulation approach

4.2.1 Simulation modelling

A simulation model is a model of some situation in which the elements of the situation are represented by arithmetic and logical processes that can be executed on a computer to predict the dynamic properties of the situation. A simulation model construction, even without absolute optimisation, is important because it results in a forward-looking point of view; it tries to provide the tools that will permit the decision maker to arrive at a better decision in the future.

In modelling even when we understand some aspects of our system better than other aspects, we should try to avoid developing a disproportionately detailed model of the familiar aspects. The accuracy of the resulting performance measured data will usually be no better than the accuracy of the performance measured in the less detailed part of our model.

A management scientist must be satisfied that he knows the system well enough to be sure that the model is valid. Without this knowledge, no amount of sophisticated programming and statistical wizardry will prevent the inevitable disaster.

4.2.2 Simulation programming

A simulation program or simulator is a computer program written to perform the simulation computations. Any sequence of clear instructions can form the basis of a computer program. Hence programs can be written which embody the logical processes which make up the system.

The creators and vendors of special purpose simulation languages will argue, quite correctly, that there is no point whatsoever in redesigning the wheel. Thousands of simulations have been programmed since the early 1960s and general principles have emerged from these experiences. We noticed that many programming languages are not designed to ease the task of logical expression. A situation for numerical optimisation is often unique in that it can not be effectively modelled using a template package or simulation language, therefore, general-purpose programming languages must be used.

The main issue of language choice is whether to program in a specially designed simulation language or whether to use a more general purpose language like FORTRAN, C, C++, or Pascal. According to Thesen and Travis (1992), three types of computer implementation of discrete event simulation are summarised in Table 4.2.3a.

Model type	Interface	User input	Flexibility & Difficulty	Typical tool
Template	Menus, mouse	Parameters	Very low	XCELL+, TBS
Simulation language	Text editor	Structure, parameters, performance measures	High	GPSS, SIMAN, SIMSCRIPT, SLAM
Programming language	Text editor	Structure, parameters, performance measures, time-keeping	Very high	C, FORTRAN, Pascal

Table 4.2.3a Three types of computer implementation

There is a growing tendency for a highly disciplined and structured approach to be taken to the programming. This is particularly important in large or complex programs. If large sums of money hang on the outcome of a computer simulation, then a professional approach is clearly necessary.

It is clear that certain specific features must be provided in any simulation, e.g., a time flow mechanism and sophisticated error messages. The latter are important because simulation programs are notoriously difficult to debug. In addition, the tasks of debugging and verification can be greatly eased if the syntax of the language employs simulation terminology and also allows the entities of the system to have meaningful names. For many reasons it may seem correct to argue that the sensible course is always to write simulation programs in a special purpose simulation language.

However, there is another point of view. It can often be more convenient to write in a general purpose language. One commonly cited reason is that the analyst may be a member of a group which already has a significant investment of time and expertise in the general purpose language. Thus there will be someone around to sort out the programs later if the analyst has moved on to another job, possibly, in another organisation. A second reason is that some of the special purpose simulation languages are suited for only some types of system and it may be easier to write better programs in the general purpose language. Often there is no need to write these programs as sets of simulation subroutines are available on a commercial basis. These carry out many of the commonly occurring tasks of a simulation.

Programming with well-defined subroutines or procedures of a manageable size is to be encouraged for two reasons:

(1) All programs need to be verified; that is, the programmer should make certain that the program accurately reflects the model; the model itself having been validated. This is easier if the program consists of modules which can be individually tested.

(2) Many simulation models grow in an evolutionary manner rather than following a precise design. This is rather easier to do in a well-structured program.

Both program design and the choice of appropriate programming languages should be considered.

4.2.3 Simulation experimentation

Computer simulation involves experimentation on a computer-based model of some system. The model is used as a vehicle for experimentation. Simulation is an experimental approach; modelling, and programming can be regarded as preliminaries to experimentation.

When a simulation model has been constructed, debugged, and validated, the analyst must design an efficient method for using it to solve the problems he had first formulated. The experiment can not be carried out without realistic data.

Some applications of computer simulation involve probabilistic components. For example, when the computer simulation involves generating values from probability distributions, it is called Monte Carlo simulation. Computer simulation may also be used when there are no probabilistic components in the model. It is natural to think in terms of adjusting the parameters of the model in order to improve the performance of the system.

When good parameter settings have been found for the model, these settings can be used to improve the performance of the real system. In some applications where a simulator has been developed, we can use the simulator to find the set of controllable variables that yields the best performance of the system. This approach is suitable for the situation where many adjustable parameters are involved.

4.3 Verification and validation

Throughout the discussion, we have made four important but unstated assumptions on simulation:

1) The model is appropriate for the decision it is intended to support.
2) The model is a correct representation of the situation being studied.
3) The model is correctly implemented.
4) The data set collected during a run is correctly manipulated and displayed.

Unfortunately, it is not possible to guarantee that these assumptions hold for any specific situation.

In order to reduce the chance of serious mistakes, verification and validation are used to make sure that the model is correctly implemented (verification) and that it is a correct representation of reality (validation).

Verification includes structured walk-through, diagnostic simulation runs, comparison of a well-understood problem, and trace analysis etc.

When a simulation model has been constructed, debugged, and validated, the analyst must design an efficient method for using it to solve the problems he had first formulated. Usually it means the two following things that are closely linked in validation.

We may first ask: would the manager of the system accept that the results of the simulation are effectively the same as those produced by his system? Firstly there is ***black box*** validity; that is, ignoring the detailed internal workings of the model, does its output accurately reflect that of the real system? In this sense, black box validity is concerned with the predictive power of the model. Does it adequately predict how the system would behave under given conditions? This is obviously a tricky question, but it must be faced. The issue of black box validity is complicated by the common fact that the simulation may be carried out because something is going wrong with the real system. Even worse, it may be a simulation of a system that does not yet exist and there is nothing with which the model may be directly compared.

We may also ask: do the components of the model represent known behaviour and/or any valid theory which exists? The second consideration is that of ***white box*** validity. One example of this is the process used to describe the arrival of customers at a queue. If the queuing system actually exists, then data may be collected which describe the arrival times of successive customers. At this point known theory can be useful. For arrival processes, certain probability distributions are known to provide a good description of the range of possible values which the inter-arrival time may take. For instance, if there is no

pattern to the arrival times then a negative exponential distribution may be appropriate - particularly if the number of potential customers is very large. Should there be no explanation of why the arrival pattern should be so random, then the suspicion is heightened. If analysis reveals that the mean and standard deviation of the inter-arrival times have very similar values then the case for accepting a negative exponential distribution is very strong indeed. This is likely to be a valid representation of the arrival process. In this case then, the arrival component of the model was verified by reference to the appropriate theory. This implies that the analyst needs to be fully conversant with the relevant theory. The same applies in the forms of simulation used in economic forecasting. Here, the aim is to develop models which show the effect of the various competing theories.

Some type of model is essential in computer simulation, the real system being mimicked by unfolding the model through time. Most managerial decision making is carried out under severe time pressure, verification and validation are something easily pushed to the back of the mind when time is short. Whatever the type of model employed, it must be valid if it is to be useful at all. This may seem obvious and so it should.

4.4 Summary

We discusseed clarification of the concept of computer simulation in this section. The method to categorise computer simulation according to major objective: problem-finding, problem-understanding; or problem-solving. Simulation approach is then studied by the stage of simulation, i.e., modelling, programming, and experimentation. We also discussed verification and validation, which are indispensible in simulation.

5. A case study through different approaches

5.1 The problem for case study

In order to illustrate optimisation, heuristic, and simulation approaches, we will present a managerial decision making problem as a concrete example. As we know, the classical steady demand no-shortage inventory control policy using the square root formula was established by 1915 and it is surprising that the analytical method has not been extended to cover cases of irregular demand. This omission has been apparent during lengthy periods of calculation (and recalculation) required by the solution of quite simple problems using dynamic programming methods. In this Chapter we will study a case - the replenishment decision making when demand increases in a linear trend. The notations used in this case are as follows:

$D(t)$ the demand rate at time t ($D(t) = a + bt$, $b > 0$);

A set-up cost, i.e., the fixed cost of a replenishment;

I stock holding cost per item per unit time;

M A/I, normalised cost;

H the time horizon;

t_i the time point of the ith replenishment;

T_i the ith time interval found by the analytic algorithm;

RC_i the cost of the ith replenishment;

RQ_i the quantity of the ith replenishment;

TC the total cost.

The following analysis might have been carried out many years ago if the time variable T_i (replenishment cycle) rather than the variable RQ_i (replenishment quantity) usually chosen as the means of expression. The cycle-time is also preferable because it facilities classification of items into groups when the inventory situation involves a large number of items. It has the technical disadvantage in extending the analysis to the case of stochastic demand (which will not be discussed here).

The instantaneous demand rate at time t is assumed as a continuous function D(t). The demand for an inventory item in the time interval (t, t+dt) is represented by D(t)dt. The replenishment quantity to be ordered at time t_i is therefore $\int_{t_{i-1}}^{t_i} D(t)dt$.

It is assumed that no shortages are allowed and that a replenishment can be made at any time point. A number of methods have been proposed for the determination of the replenishment policy for a product where demand is increasing linearly. Without loss of generality, we assume that the planning horizon H is finite. The initial and final inventory levels are zero. In practice, demand will not usually cease at the horizon, the solution will be sub-optimal, and the procedure would be to review the situation and calculate new t_is at some point before the horizon is reached.

Referring to a replenishment policy, $0 = t_0 < t_1 < \ldots < t_i \ldots < t_{n-1} < t_n = H$, the objective of this managerial decision making problem is to minimise the total replenishment cost TC, i.e.,

$$\min TC = \sum_{i=1}^{n} RC_{i-1}$$

where

$$RC_i = A + I\int_{t_{i-1}}^{t_i} (t-t_{i-1})D(t)dt = A + I[\,(1/2)D(t_{i-1})T_i^2 + (b/3)T_i^3]$$

is the replenishment cost for replenishment at time t_i for time period $T_i = t_i - t_{i-1}$.

We will study this case through various methods. As an example of this algorithm, let the demand $D(t) = 900t$, and $H = 1$, $A = 9$, $I = 2$.

5.2 An optimisation approach

Donaldson (1977) developed an analytical optimal algorithm for the replenishment policy when there is a linear trend in demand within a time horizon. Using methods of calculus a computationally simple procedure for determining the optimal times for replenishment of inventory is established.

The procedure can be described as the following two cases:

Case A (D(t) = a+bt with a = 0)

Step 1. Determination of the optimal number of replenishment intervals - n.

a) Evaluate $K = M/(H^3 b)$;

b) The optimal value of n is that for which

$g(n) > K \geq g(n+1)$,

where

$g(n) = G(n) - G(n-1)$, $n \geq 2$,

where

$$G(n) = \begin{cases} (1/g_n^3)\sum_{i=1}^{n-1} g_n^3 (1-1/Z_i) & n \geq 2 \\ 0 & n = 1 \end{cases}$$

where

$$g_i = \begin{cases} \prod_{j=2}^{i} Z_j & i \geq 2 \\ 1 & i = 1 \end{cases}$$

and

$Z_1 = \infty$,

$Z_{i+1} = (3-2/Z_i)^{1/2}$, $\quad i = 2,\ldots,n$

Step 2. Determination of the optimal replenishment times.

a) The initial value is set as

$t_0 = 0$; and $t_1 = H/g_n$.

b) The analytic procedure is given as follows:

$t_i = (g_i/g_n)H, \qquad i = 2,...,n-1.$

Case B (D(t) = a+bt with a ≠ 0)

Step 1. Solve the associated problem with demand rate D(t) = bt and time horizon H+a/b. By using the above procedure in Case A and find the optimal number of replenishments n*.

Step 2. Determine a non-zero replenishment point r such that $t_r \leq a/b \leq t_{r+1}$.

It follows that the optimal number of n for the original problem is either n*-r or n*-r-1.

Step 3. Determine optimal replenishment points for each possible values of n.

a) With given equations

$$Z_{i+1} = (3-2/Z_i)^{1/2}, \qquad i = 2,...,n$$

and

$$\prod_{i=1}^{n} Z_i = 1 + (b/a)H,$$

determine Z_1 and then Z_i for i = 2,..,n.

b) According to the above $\{Z_i\}$, the replenishment points are then given by

$$t_i = (a/b)(\prod_{j=1}^{i} Z_j - 1).$$

Step 4. Evaluate the two values of the TC to choose the optimal policy.

Since the optimisation algorithm is not only lengthy but also complex in nature, it is not surprising a number of researchers have proposed a variety of different heuristics.

5.3 Heuristic approaches

5.3.1 Examples of problem substitution heuristics

a) Knowledge-based heuristics

In the above case, one of well-known heuristic algorithm is the continuous time part period algorithm (CTPP). It is based on the principle of choosing lot sizes so as to equalise set-up and inventory cost:

$$A = I\int_{t_{i-1}}^{t_i} (t-t_{i-1})D(t)dt$$

This heuristic method ignores the original optimisation problem, and simply relaxes the optimisation model to an equation. The numerical optimisation problem is replaced by an iteration problem.

It is easy to show that the necessary condition for the CTPP can be derived as

$$T_i^2[D(t_{i-1})/2+(1/3)bT_i] = M,$$

and we then have the following result (Table 5.3.1a):

No.	Replenishment Point t_{i-1}	Replenishment Cost RC_{i-1}	Replenishment Quantity RQ_{i-1}
1	0.0000	18.0013	27.3725
2	0.2466	18.0000	49.6639
3	0.4138	18.0000	61.1491
4	0.5541	18.0000	69.5144
5	0.6794	18.0000	76.2720
6	0.7944	18.0000	82.0223
7	0.9018	17.3876	84.0058

Table 5.3.1a Computational results obtained by CTPP

The total replenishment cost is 125.3889, and the number of replenishments is 7.

b) Common sense heuristic

It is common sense to reduce unit cost so as to reduce the total cost. In the above case, another well-known heuristic algorithm is the Continuous Least Unit Cost method (CLUC). It selects the duration of the replenishment interval at each replenishment, which

minimises the relevant cost per item over this interval. Consequently, the optimisation model can also be replaced by

$$\min \{(A+I\int_{t_{i-1}}^{t_i} (t-t_{i-1})D(t)dt)/\int_{t_{i-1}}^{t_i} D(t)dt\}$$

This heuristic method is straightforward: to minimise the unit cost per period so as to reduce the total cost.

Similarly, the necessary condition for CLUC can be then derived as follows:

$$T_i^2[D(t_{i-1})/2+(1/6)bT_i] = M$$

and we have (Table 5.3.1b):

No.	Replenishment Point t_{i-1}	Replenishment Cost RC_i	Replenishment Quantity RQ_{i-1}
1	0.0000	27.0026	43.4512
2	0.3107	19.3557	58.5369
3	0.4761	18.7950	67.9093
4	0.6145	18.5651	75.1611
5	0.7380	18.4390	81.2049
6	0.8515	18.3592	86.4483
7	0.9577	10.5897	37.2884

Table 5.3.1b The computational results obtained by CLUC

The total cost is 131.1063, and the number of replenishments is 7.

5.3.2 Examples of model substitution heuristics

a) Relaxation on the objective

An adaptation of the Silver-Meal heuristic selects the next replenishment interval by minimising the relevant cost per unit time over the duration of the replenishment interval.

For any given replenishment point t_{i-1}, the next replenishment point t_i can be determined by the following optimisation model

$$\min (A + \int_{t_{i-1}}^{t_i} (t-t_{i-1})D(t)dt)/(t_i-t_{i-1})$$

That is, the heuristic method iterates the optimal solution of the relevant cost per unit time.

The necessary condition for Silver's model can be derived as

$$T_i^2[D(t_{i-1})/2+(2/3)bT_i] = M,$$

and we have (Table 5.3.2a):

No.	Replenishment Point t_{i-1}	Replenishment Cost RC_i	Replenishment Quantity RQ_{i-1}
1	0.0000	13.5006	17.2436
2	0.1958	15.6739	38.7817
3	0.3528	16.4672	51.8208
4	0.4895	16.8665	61.3428
5	0.6132	17.1043	68.9497
6	0.7275	17.2611	75.3476
7	0.8346	17.3720	80.9091
8	0.9362	12.5877	55.6047

Table 5.3.2a The computational results obtained by Silver's

The total replenishment cost is 126.8333, and the number of replenishments is 8.

b) Relaxation on horizon

Back to our example, Ritchie (1984) modified Donaldson's optimisation method, for a linear increasing demand function with zero initial demand, by noting that a ratio, used by

Donaldson for calculating the first replenishment interval, approached a constant, 0.43, as the number of replenishment intervals approached infinity.

Tsado (1985) extended Ritchie's result for demand functions with a non-zero initial demand, and proposed a cubic equation to give the duration of the replenishment.

With that constant, Ritchie's cubic equation is given as follows

$$T_i^2[D(t_{i-1})/2+0.43bT_i] = M,$$

and we have (Table 5.3.2b):

No.	Replenishment Point t_{i-1}	Replenishment Cost RC_{i-1}	Replenishment Quantity RQ_{i-1}
1	0.0000	15.9757	23.0942
2	0.2265	17.2214	45.8176
3	0.3913	17.5245	58.0381
4	0.5311	17.6592	66.8570
5	0.6563	17.7349	73.9231
6	0.7713	17.7833	79.8992
7	0.8789	17.8168	85.1228
8	0.9806	9.3349	17.2480

Table 5.3.2b The computational results obtained by Ritchie's

The total replenishment cost is 131.0506, and the number of replenishments is 8.

5.3.3 Examples of algorithm substitution heuristics

a) An eclectic heuristic

An interesting result is that the necessary conditions of the above 4 heuristics have the same mathematical structure, and differ only by a single parameter in an equation although these cubic equations are derived from different considerations. That is,

$$T_i^2[D(t_{i-1})/2+\lambda bT_i] = M.$$

The parameter λ is then considered as adjustable and its original economic explanation is clear only if $\lambda \in \{ 0.43, 2/3, 1/6, 1/3 \}$.

Amrani and Rand (1990) used an iterative method to solve the cubic equation, and then chose the best replenishment policy which minimises the total replenishment cost. The existence of a real positive solution of the cubic equation can be proved strictly, the convergence rate of an iteration method is not satisfactory and even fails in some cases. When compared with the results in Yang and Rand (1993), computational results show that rounding errors were present when using the iterative method.

b) An approximation heuristic

Amrani and Rand (1990) proposed an algorithm to solve the cubic equation. An iterative method was proposed to find the right solution out of three roots of a cubic equation. Let T(k) be the value of T_i obtained on the k*th* iteration, then the cubic equation can be rewritten as

$$T(k+1) = [\ 2M/(\ D(t_{i-1}) + 2\lambda T(k)\ )\ ]^{1/2}$$

Initially, set $T(0) = 0$ when $D(t_{i-1}) > 0$, normally T(k) is approaching to a positive root of the cubic equation.

The disadvantages of an iterative algorithm are that the convergence rate and indeed convergence itself can not be guaranteed, and moreover, the computation error accumulates as the number of replenishments increases.

ASTON UNIVERSITY
LIBRARY AND
INFORMATION SERVICES

c) **An analytic heuristic**

Yang and Rand (1993) proposed an analytic algorithm for the cubic equation. Therefore, the above iteration heuristic can be regarded as an approximation. As indicated before, the key requirement is to choose the particular root of a cubic equation. There is not a general algebraic analytic formula for the roots of a polynomial equation up to power 5 or more, implied by the Abel Theorem.

For the cubic equation, if the adjustable parameter $\lambda > 0$ and $b > 0$, the existence of a unique positive root can be proved based on the mathematical properties of this common form, especially the distribution of the three roots of the cubic equation.

Step 0. Initially, set the starting time t_j and a parameter $\lambda > 0$.

Step 1. Computer the following values

$$U = D(t_{j-1})/(2\lambda b), \qquad V = -M/(\lambda b);$$

$$Q = -U^2/9, \qquad R = -(27V+2U^3)/54;$$

and then the discriminate

$$\Delta = Q^3+R^2.$$

Step 2. If the discriminate $\Delta > 0$, then go to Step 3; otherwise, go to Step 4.

Step 3. Since it can be shown that if $\Delta > 0$, the other two roots are conjugate imaginary; set

$$S_1 = \text{Sign}(R+\Delta^{1/2})\ \text{Abs}(\ R+\Delta^{1/2}\ )^{1/3},$$

$$S_2 = \text{Sign}(R-\Delta^{1/2})\ \text{Abs}(\ R-\Delta^{1/2}\ )^{1/3};$$

where Sign (.) is sign function and Abs(.) is the absolute value function. So the solution required is

$$T_i = S_1+S_2-U/3,$$

and then stop.

Step 4. Since it can be shown that if $\Delta = 0$, the other two roots are negative and equal; and if $\Delta < 0$, the other two roots are negative, set

$$\theta = \arccos(R/(-Q)^{3/2}),$$

and

$$T^a = 2(-Q)^{1/2}\cos(\theta/3)-U/3,$$

$$T^b = 2(-Q)^{1/2}\cos(\theta/3+2\pi/3)-U/3,$$

$$T^c = 2(-Q)^{1/2}\cos(\theta/3+4\pi/3)-U/3;$$

hence, the solution required

$$T_i = \text{Max}\ \{\ T^a, T^b, T^c\ \},$$

and then stop.

Needless to say, this analytic algorithm can solve the cubic equation with greater accuracy and less computation time than the previously proposed iterative method.

d) An improvement heuristic

It is believed that much of the cost penalty arises from the fact that the replenishment intervals do not coincide exactly with the time horizon H. The time of final replenishment usually has to be adjusted to ensure coincidence with the time horizon H.

An alternative way to deal with the final replenishment, when the interval (k_{n-1}, H) is too narrow, is to reduce the total cost by cancelling the (n-1)th replenishment. Then the final replenishment cost is,

$$RC'_{n-2} = A+I((1/2)D(t_{n-2})(H-t_{n-2})^2 +(1/3)b(H-t_{n-2})^3],$$

instead of $(RC_{n-2} + RC_{n-1})$.

Obviously, if

$$(RC_{n-2} + RC_{n-1}) > RC'_{n-2},$$

then final replenishment point t_{n-2} is meant by the new total cost

$$TC = \sum_{j=1}^{n-3} \Sigma RC_j + RC'_{n-2}.$$

This is also a kind of algorithm substitution heuristic.

5.4 A computer simulation approach

a) The simulation model

The recognition of this common structure allows not only an eclectic heuristic approach, which simply calculates the results for all four heuristic methods, but also a search procedure. According to an analytic procedure, some other replenishment plans can be obtained by adjusting this parameter with less computation time.

Based on this consideration, Yang and Rand (1993) present a generalised eclectic model:

Minimise { $TC(\lambda)$: $\lambda > 0$ }.

Where TC(.) is a mapping determined by the analytic procedure in Yang and Rand (1993). Since we can find the right root of the equation quickly and accurately by the analytic procedure, it is more practical to minimise the total cost TC by simulation with a large number of values of the parameter λ.

b) The simulation program

A program written in Turbo-Pascal is attached in the Appendix.

c) The experimentation

Referring to the well-known problem a = 0, b = 900, H = 1, A = 9, and I = 2, Yang and Rand (1990) simulate this replenishment policy with $\lambda \in S2 = \{i/100 : i = 1,...,99\}$, each parameter determines a set of replenishments, as well as the total cost.

Figure 5.4a illustrates the total cost TC, and Figure 5.4b illustrates the number of replenishments.

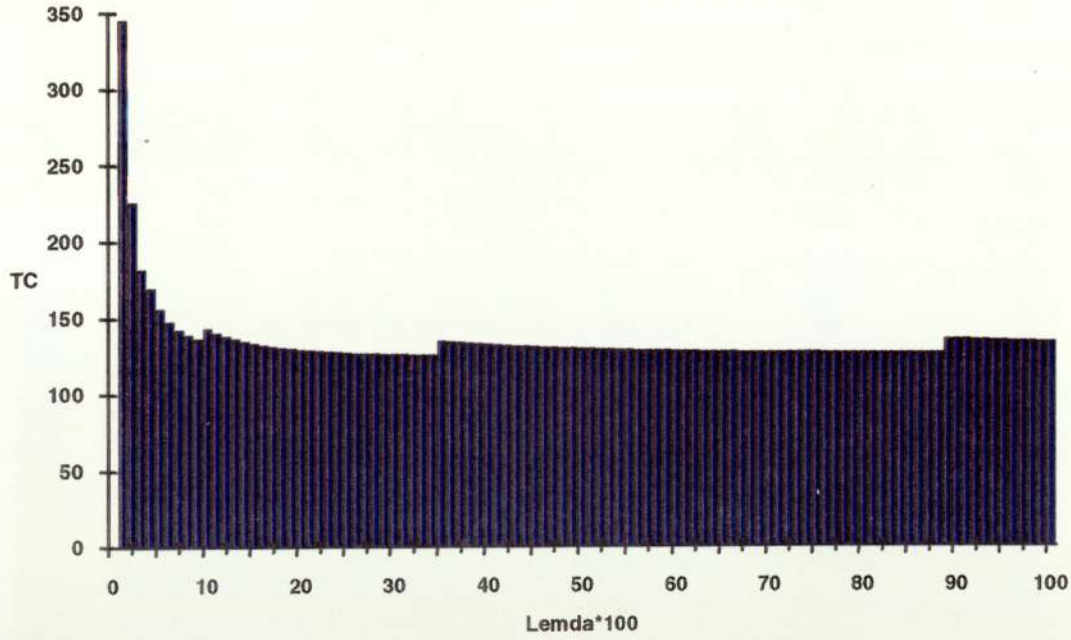

Figure 5.4a The total cost v. parameter λ

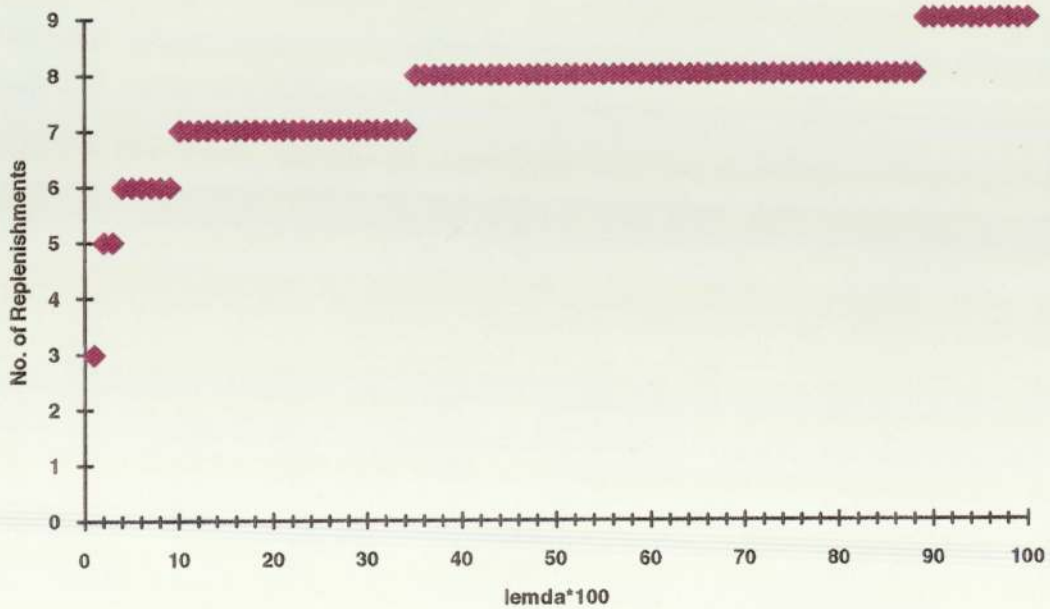

Figure 5.4b The number of replenishments v. parameter λ

Compared to the optimization approach and heuristics, the simulation approach can provide more information about the problem. According to above figures, we can see that

(1) if the number of replenishments is fixed, the total cost TC is more likely to be a decreasing convex function;

(2) it is not always a continuous one, because the final replenishment is determined;

(3) the larger λ is, the greater will be the number of replenishments.

Better performance of such a decision making pattern can be found by more complicated computer simulation. For example, for $\lambda = i/1000$ where $i = 1,...,1000$, the more satisfactory result is found when $\lambda = 0.3460$. That is (Table 5.4b).

No.	Replenishment Point t_{i-1}	Replenishment Cost RC_{i-1}	Replenishment Quantity RQ_{i-1}
1	0.0000	17.6718	26.7005
2	0.2436	17.8940	49.1177
3	0.4105	17.9370	60.7170
4	0.5508	17.9552	69.1485
5	0.6761	17.9653	75.9501
6	0.7911	17.9717	81.7321
7	0.8986	17.9412	86.6341

Table 5.4b The computational results obtained by computer simulation

The total cost is 125.3362, and the number of replenishments is 7.

Compared with the previous eclectic algorithm, the percentage increase in total cost above the optimal is reduced from 0.1029% to 0.06%. By using this simulation approach, a balance can be easily struck between the number of replenishments and the cost penalty. For example, we can compare the above result when $\lambda = 0.3460$ with that obtained when $\lambda = 0.0970$: the number of replenishments can be reduced from 7 to 6 at a cost penalty of 9.0377 (= 134.3739-125.3362).

The simulation can generate many alternative replenishment plans: some of which may be more practical. For instance, if a certain period is not suitable for replenishment, we can choose a replenishment plan by adjusting the parameter, to minimise the total cost with the constraint.

5.5 A comparison on 12 problems

For an inventory problem with a finite planning horizon and zero initial and final inventory level, different methods can be compared on a range of problems. The result

depends on the demand rate, set-up cost, stock holding cost per item per unit time, as well as the time horizon. One may argue that one specific problem can not be representative.

Different heuristics can be tested in a specified problem so as to check the efficiency. The results of simulation on a group of problems can indicate which heuristics perform the best. In order to compare the above algorithms, the following 12 sample problems have been tested(Table 5.5).

No.	a	b	H	A	I
1	0	900	1	9	2
2	0	900	2	9	2
3	0	100	4	100	2
4	0	1600	3	42	0.56
5	6	1	11	30	1
6	6	1	11	50	1
7	6	1	11	60	1
8	6	1	11	70	1
9	6	1	11	90	1
10	100	150	1	30	2
11	100	150	1.5	30	2
12	100	150	2	30	2

Table 5.5a. The parameters of the sample problems

We suggest a less time-consuming simultion approach by searching

$$\lambda \in S1 = \{i/10 : i = 1,...,9\},$$

instead of

$$\lambda \in S2 = \{i/100 : i = 1,...,99\}.$$

The computational results are different from Yang and Rand (1993) by column S1(Table 5.5b.).

Problem	The optimal	% above the optimal						
No.	Donaldson's	Ritchie's	Silver's	CLUC	PPA	Ecl.	S1	S2
1	125.26	4.6229	1.2555	4.6673	0.1029	0.1029	0.3319	0.0608
2	345.78	0.5959	2.1104	1.3381	1.6275	0.5959	0.0469	0.0463
3	1122.60	0.2016	5.3479	10.0957	1.3820	0.2016	0.0158	0.0155
4	977.17	0.0357	1.5876	5.4091	0.6230	0.0356	0.1319	0.0029
5	291.21	0.0258	7.6566	1.0968	0.2085	0.0258	0.0619	0.0019
6	378.05	0.1748	11.7907	2.0514	0.5622	0.1748	0.0414	0.0043
7	418.05	2.2302	0.5930	6.6906	3.4220	0.5930	0.0586	0.0307
8	450.84	7.3633	4.2506	14.3612	9.3362	4.2506	0.2269	0.0907
9	510.84	0.0690	14.4429	2.0304	0.3953	0.0690	0.0017	0.0010
10	150.42	2.7269	0.8682	7.8399	4.0666	0.8682	0.1534	0.0176
11	242.46	9.1443	5.4776	0.4576	11.3779	0.4576	0.2626	0.0100
12	347.64	0.0529	6.0923	1.7656	0.3529	0.0529	0.0018	0.0005
Overall		1.2601	4.8847	5.6409	2.0181	0.5418	0.1111	0.0171

Table 5.5b The computational results obtained by various algorithms

The analytic eclectic algorithm, just using the four heuristics, reduced the average percentage above the optimum down to 0.5418%. However, the improved eclectic S1 reduces it again to 0.1111%, and S2 finally reduces the average result to only 0.0171%.

A particular heuristic may have a very poor behaviour caused by rare values of some parameters, yet perform excellently under most other conditions. For example, Silver's heuristic gives a good result in problem 7, while giving a poor result in problem 9. One may even suggest that the simulation might find an optimal solution if all feasible values for parameter λ had been tested. However, we have pointed out that this approach is still a heuristic algorithm unless it can be proved mathematically.

6. Extensive discussion on the three approaches

6.1 Methodology consideration

Structured approaches to numerical optimisation problems have generally evolved along three approaches: optimisation, heuristics, and simulation. All have been used as an integral part of decision support tools. Each offers valuable characteristics and uniqueness.

a) Optimisation methodology

Optimisation is potentially the ideal way to solve a decision problem. The problem is represented by means of mathematical expressions, and then the best alternative is found through the application of mathematical logic.

This mathematical logic is embodied in such well-known procedures as differential calculus and mathematical programming. However, this ideal approach exacts its price. Because guaranteeing that the best solution will be found can require significant computer running time and memory, real-world problem descriptions frequently must be approximated and abbreviated. That is, problem descriptions can rarely be as extensive and in as much detail using an optimisation approach as they can be for a simulation approach.

Problem description detail must often be sacrificed for the capability to find the optimal solution. The danger, of course, is finding an optimal solution to a problem description that is not sufficiently close to reality to be convincing, or of experiencing computer running times so long as to make this approach impractical. Some attempts have been made to develop special approaches that take advantage of specific problem structures in order to overcome some of these disadvantages.

b) Heuristic methodology

Well-known heuristic decision rules may not only have a sound basis in economic or mathematical theory, but are evidently the best way to incorporate economic logic into a stream of fast-moving decisions.

It is easy to examine an optimisation algorithm since there is only one short cut to Rome. However, it is difficult to examine heuristics systematically since "all roads lead to Rome". These rules that guide the solution-finding process abound in problem solving. They are not difficult to specify because researchers are in the habit of using them constantly in daily activities.

Good heuristics are frequently the result of common sense procedures that work effectively. They may be based on concepts, principles, and theories that relate to a specific problem, or they may result from observing the form of optimised solutions and mimicking them. It is not easy to recognise when good heuristics have been found, since testing them against other methods is difficult when practical-size problems are involved.

The use of heuristics in solving problems attempts to maintain the level of problem description detail of simulations while offering the best solution search capability of optimisation approaches. Simply put, heuristics are rules of thumb that direct the solution approach toward the best solution, but do not guarantee that it will be found more specifically.

A heuristic is a short cut process of reasoning that searches for a satisfactory, rather than an optimal solution. The heuristic, which reduces the time spent in the searches for a satisfactory, rather than an optimal solution, comprises a rule or a computational procedure which restricts the number of alternative solutions to a problem, based upon the

analogous human trial-and-error process of reaching acceptable solutions to problems for which optimising algorithms are not available.

The performance of this approach depends a great deal on the quality of the heuristics used. Quality heuristics would allow optimal or near optimal solutions to be found in a fraction of the computational time required for optimising approaches. The resulting savings in computational time, and corresponding memory, can be used to provide more realism in the problem description. Finding such good heuristics can be elusive and it is often difficult to show that they work well in realistically-sized problems.

c) Computer simulation methodology

Computer simulation is a mathematical description of a decision problem, usually in significant detail. The mathematical description is typically manipulated with the aid of a computer due to the burdensome computations required. However, it is the extent of this problem description detail that distinguishes simulation from other decision approaches.

Problems are solved by "costing out" various alternatives as replicated by the simulation. Repeating the simulation numerous times produces a cost profile for the various alternatives from which the most desirable one may be selected. Although simulations may be written in general programming languages (FORTRAN, C, C++, BASIC, and PASCAL), special-purpose languages exist to facilitate simulation development (SLAM, SIMSCRIPT, and GPSS). Some simulation programs are prewritten to solve specific problem types (SIMFACTORY and LREPS).

A characteristic of detailed simulations is that they require substantial computer running time, especially if the problem is replicated in great detail. They also place the burden on the user to seek out the best alternative to test.

We do not discuss which methodology is better for numerical optimisation problems in managerial decision making. Throughout the rest of this Chapter we use notation "$\propto$" for "is not better than".

6.2 Modelling consideration

Algorithms are applied to concrete model structures. Modelling effort is an important factor in the choice of a suitable approach. We attempt to conclude some modelling criteria for evaluating the different approaches.

a) Model modification

It is often necessary to modify a specific problem or model so as to apply known algorithms to them. Optimisation algorithms require modification. Heuristics normally need no modification. However, simulation requires a lot of modification.

Our judgement: Simulation $\propto$ Optimisation $\propto$ Heuristics.

b) Program development

In managerial decision making it is almost impracticable to apply an algorithm without computer or program in this IT revolution time. One such class of procedures is computerisation. To be considered an effective alternative to model based decision making effort, a good model must possess the following features: substantial simplicity, reasonable computer storage requirements, reasonable computing time and cost, acceptable accuracy and validity of solutions, robustness, generation of multiple solutions, and user friendliness.

One is reluctant to develop a program when optimisation software is available. A standard optimisation algorithm normally does not require program development. A non-standard

optimisation algorithm requires program development, which is often more complicated and often requires much more effort than heuristics.

Heuristics normally need program development, but, it is usually not difficult to develop a program for a heuristic algorithm. Compared to simulation, heuristics require less effort in developing a program.

For a simulation algorithm, however, program development is usually a substantial undertaking. Realistic simulations often require long computer programs of some complexity. There are special purpose simulation languages and packaged systems available to ease this task, but it is still rarely simple. For a particular problem, one should certainly not attempt to develop a simulation language.

Our judgement: Simulation $\propto$ Optimisation $\propto$ Heuristics.

c) Data processing

Some industrial problems are so complex that the standard models of operational research are inappropriate. This sometimes occurs because: 1) the number of pieces of information needed to describe the problem is enormous; 2) the problem has features which are difficult to quantify or involve a conflict of objectives; 3) it may be difficult to collect accurate data (Foulds, 1983).

If an optimisation algorithm is supported by commercial optimisation software then data processing is not an easy task. However, data processing may be easier if the optimisation algorithm is programmed for a specific problem.

Heuristics are often more flexible in data processing whether supported by a program or not. A heuristic program is normally developed for a particular problem, which does not

have to transform crude data into specified parameters, but uses crude data directly. Therefore, a heuristic algorithm reduces the chance of information distortion.

Similarly, the design of a simulation program often takes data processing into account. which means that computer simulation normally requires simpler data processing.

Our judgement: Optimisation ∝ Simulation ∝ Heuristics.

6.3 Feasibility consideration

We provide no comfort whatsoever when telling a manager that there does not exist an efficient method for finding a solution. However, Privately we think hard about the technical feasibility of all the possible approaches. We attempt here to conclude some feasibility criteria for evaluating the different approaches.

a) Optimality

The word optimality here means the difference between the optimal solution and the solution found by one of these approaches. However, it may be indicated by the difference between the values of the objective. With the support from optimisation software, the optimisation approach is still a powerful tool for standard OR/MS problems where optimality has a high priority.

An optimisation algorithm can guarantee to find an optimal solution for any instance of the problem.

A heuristic method cannot guarantee to find an optimal solution at least in one instance of the problem. Different heuristics may be suitable for different situations of a problem. Optimality of a heuristic can be checked by the least upper bound according to the conditions in a specified problem. In model relaxation with bound, we have briefly discussed the implication of bound. A least upper bound can be used as a criterion when

comparing a group of heuristics. Certainly, heuristics are no panacea. A heuristic solution would appear doubtful to managers when it is impossible to estimate its closeness to the optimal (even if it exists).

A computer simulation cannot guarantee to find or converge on an optimal solution. However, a trial-and-error approach can find the best solution. The word best in simulation may be somewhat misleading. It does not mean the optimal solution in any absolute sense, but refers instead to the best solution that the decision maker can attain with the resources and time available. One usually selects those values of the parameters to test in the model that have a good chance of being near the optimal solution.

Our judgement: Heuristics $\propto$ Simulation $\propto$ Optimisation.

b) Complexity of algorithm

Computer experts often use the term polynomially-bound time, as it is assumed that computational time is linearly proportional to the number of elementary computational steps. The size of a specified instance of a problem is defined to be the number of the symbols required to describe it. So-called P or NP become very important in evaluating an algorithm, especially for a combinatorial problem. An algorithm is considered to be *effective* if it can guarantee to solve any instance of the problem for which it was designed by performing a number of elementary computational steps where this number can be expressed as a polynomial function of the size of the problem.

An optimisation algorithm is often of higher complexity than other algorithms. Many numerical optimisation problems in managerial decision making fall into NP-hard problems such as capacitated lot-sizing, and machine-scheduling, etc. This implies that no polynomial optimisation algorithm can be possibly found.

A heuristic algorithm is simple and effective, therefore non-NP is being considered in the first place.

A computer simulation is often very time-consuming and therefore is no longer judged by the complexity of its algorithm.

Our judgement: Simulation ∝ Optimisation ∝ Heuristics

c) Sensitivity analysis

When using forecasts to assist in decision making, we often try to protect our solution from forecasting errors by obtaining a measure of the sensitivity of the solution.

It is much easier to undertake sensitivity analysis using an optimisation algorithm.

Sensitivity analysis can also measure the effects on the heuristic solution, however, it is not straightforward.

In computer simulation it is difficult to undertake sensitivity analysis because it is a trial and error approach.

Our judgement: Simulation ∝ Heuristics∝ Optimisation

6.4 Applicability consideration

In practical OR, the hard task is to discover precisely what the real problem is. Then, either an adequate solution can be found without detailed calculations, or alternatively some new option must be envisaged, and this requires inspiration rather than calculation. Different models can also help decision-makers in new situations. The decision problem may well change significantly before a satisfactory solution is produced by a suitable

approach. An approach that survives and is widely used will have to be user friendly and have other good qualities for solving numerical decision making problems. The only accepted measure is whether it has a strong applicability. We attempt to conclude some applicability criteria for evaluating the different approaches.

a) Computational time

The computational time of an algorithm can be tested on a specified sample or a group of samples. It owes as much to better mathematics as to better computers.

An optimisation algorithm normally requires reasonable time whenever a low convergence rate occurs in some optimisation algorithms.

Heuristics often require little time since the design of a heuristic algorithm takes convergence rate into account.

Searching for an optimal solution from a computer simulation can turn out to be a surprisingly time-consuming process.

Our judgement: Simulation $\propto$ Optimisation $\propto$ Heuristics

b) Managerial preference

The intuition, experience and common-sense of the manager remain indispensable for the selection of an alternative solution and its implementation. Nevertheless, there is something strange about that intuition: sometimes the optimal solution may appear not to be the "best", when compared with the heuristic or simulated solution. Managers require different alternatives for one reason or another, the more alternatives, the better.

An optimisation algorithm normally determines merely one optimal solution, e.g., some convexity condition is satisfied.

A heuristic method also often determines one solution. However, since heuristics are simpler it is possible to produce more alternatives according to different considerations.

It can be seen that simulation allows managers to explore the whole range of feasible options in a decision problem. These options could not be explored without a powerful computer, even though the selection process would be very slow.

Our judgement: Optimisation $\propto$ Heuristics $\propto$ Simulation

*c) **Maintenance effort***

The effectiveness of maintenance directly affects the quality of managerial decision making, which is a relatively neglected part of OR. For example, when a series of similar products are being produced to order using the same technology, the requirement for maintenance is minimal. However, when the technology is changing rapidly, constant maintenance becomes necessary.

Without expert help from OR professionals, the maintenance of optimisation software is difficult whenever there is a change in the managerial problem.

Heuristic packages normally suit actual data well so that the maintenance of such a package would be much easier.

Maintenance of a simulation system is far more difficult or even impossible.

Our judgement: Simulation $\propto$ Optimisation $\propto$ Heuristics

7. Major conclusions and summary

7.1 Major conclusions

The current tendency in decision making shows that companies today have to take an in-depth look at these approaches which propel them toward a higher level. Any OR/MS professionals may note that it is difficult to examine these approaches systematically.

We attempt to conclude the following eight points on the basis of this research:

1) No evidence can proved breakthrough in the use of optimisation algorithms and theory

It can be seen that apart from a certain class of traditional management problems which is solvable using standard optimisation algorithms, there is a strong possibility pure optimisation will not be undertaken extensively for managerial decision making. But there is no evidence that optimisation has made a breakthrough in the major areas of optimisation such as linear programming (LP), non-linear programming (NLP), and large-scale programming (LSP). In Chapter 2 and appendix 1, it can be shown by the fact that most available optimisation software was still designed by classical optimisation theory and methods, e.g., varied simplex algorithms for LP, and modified Lagrangean or Newton algorithms for non-linear programming etc. We can conclude that it is unlikely that some breakthrough had been made for optimisation on the basis of traditional linear programming and non-linear (convex) programming, otherwise, we should have very powerful optimisation software designed by new algorithm.

2) The simplicity consideration is most important in managerial decision making

On the basis of the survey at British Airway, we found heuristics are proposed for simplification through a kind of substitution in problem, model, or algorithm. This helps us to classify heuristics and study them in depth. Relaxation of optimality requirements is the central idea of heuristics. Decision making is always an on-going process so that

managers may not describe their problem well in the first place. Over-enthusiastic effort becomes unnecessary or does not match a roughly defined managerial decision making problem. In many situations, a heuristic algorithm is more suitable to be used than an optimal algorithm. We can conclude that the simplicity of the heuristic approach makes it an attractive decision support tool for managers because they are more interested in decision analysis, rather than decision making.

3) Decomposition and heuristics together may solve large-scale problem

According to the study on decomposition, we found early experiences may have left a generally negative and misleading imprint on the entire approach due to lack of extensive and systematic studies of the behaviour of decomposition algorithms. For well-behaved LP models, the convergence as measured by the number of times the master problems must be resolved, is actually surprisingly fast. Slow convergence may be caused, at least in part, by the propagation of numerical errors. There are indeed meaningful, large-scale optimisation applications that eventually may have to rely on decomposition. Theoretical results including principles are leading to mathematical representation, and then more robust algorithms. New computer architecture allowing parallel computation will provide further opportunities to realise the potential of the decomposition approach. The prospect of decomposition using heuristics may also have significant impact on many real-world applications.

4) Time restriction prevents managers from using optimisation approach

It goes without saying that simulation is the most unpopular approach according to time restriction. However, it can be seen optimisation approach often fails as well. As we know, managers would like to examine a managerial problem in breadth rather than in depth because of time restriction. Many known managerial problems are so-called NP problems, i.e., there does not exist a P algorithm for such a problem. Once an algorithm falls in NP-hard or NP-complete, the implication is that the number of operations or computation time will increase sharply with number of variables. In this situation,

computation time becomes non-practical for large-scale or even medium size problems. It is therefore not surprising that a variety of heuristics has been suggested for solving optimisation problems.

5) Data formality prevents managers from using commercial optimisation software

As indicated before, there are large number of commercial optimisation software available. But there are few reports on their applications in decision making. In this research we found commercial optimisation software requires rigid data formality, which implies considerable modelling effort to transfer crude data to standard data or specified parameters. Such a transformation may lose or distort some characteristics of the original information. Therefore, optimisation software is restricted to standard decision making problems. However, heuristics do not require much data formality because they are proposed for specific problems.

6) Correct recognition of the position of OR/MS professionals is required badly

An optimisation technique is used to analyse a managerial problem, not actually to make a decision. OR analysts are actually decision making assistants for a managerial problem, not decision makers. Optimisation algorithm often provide managers with a unique solution. Managers have to raise many questions about this solution, which may not be answered by an optimisation model. If there are two or more objectives then the costly optimisation algorithm will not guarantee a more satisfactory solution for managers. Flexible heuristics can often serve for decision analysis purposes, while optimisation algorithms and computer simulation imply formidable tasks that are not always worth undertaking. This point of view also came from the interview with the key people in the OR group in British Airways.

7) Problem-finding possibilities exist in heuristic and simulation approaches

On the basis of the research on heuristics, we found heuristics are much more flexible in dealing with decision making problem. In a case study we studied a number of heuristics,

and found each of them has some special characteristics. Some problem substitution heuristics have a clear economic background and can be understood easily. Compared to heuristics and simulation, in reality an optimisation algorithm has no more importance for such a managerial problem. Managers have a potentiality to try these heuristics or even make some heuristics themselves. Furthermore, managerial decision making problems are often multiple criteria by nature although they are modelled as single objective numerical optimisation problems. Optimisation algorithm is basically a problem-solving approach, while the heuristic approach is in nature both problem-finding and problem-solving. If a managerial decision making problem is different from those classical or standard OR problems, it often requires problem-finding which is more likely to be met by a heuristic approach.

8) The influence of IT development will make all the approaches more powerful

The spread of personal computers is favourable for heuristics, which will become more and more important in decision making with the development of information technology. Most managers have some knowledge about spreadsheet and database because new generation of managers are well prepared. However, they will benefit more from new I.T. such as data processing rather than optimisation software, which still needs OR/MS expertise. The development of information technology increases the applicability of the these approaches, especially simulation. These approaches will undoubtedly be built around computer-based decision support systems. The growing number of well-designed DSS systems in the industry may promote the use of quantitative heuristic decision techniques.

We compared the three approaches through methodology, modelling, feasibility, and applicability. Heuristic provides a way of quickly finding satisfactory solutions to problems when such methods as simulation and optimisation prove undesirable or impracticable. Simple, understandable and usable heuristic approaches for solving managerial decision making problems are needed. It will be more attractive to managers

in the future with the development of IT. Traditional optimisation is sometimes inadequate, but decomposition techniques may be promising in dealing with practical problem, especially combined with heuristics. Simulation also has its potential with the development of information technology. Computer simulation then may well be regarded as the last resort. Despite this, it is surprising how often such an approach is needed. There are certain advantages in employing a simulation approach in management science and it may be the only way of tackling some managerial decision making problems.

7.2 Summary

This research indicates that heuristics are more suitable for managerial decision making. It provides a way of quickly finding satisfactory solutions to problems when such methods as simulation and optimisation prove undesirable or impracticable. It is concluded that operations management should consider integrating these heuristic methods into a decision support system. Diversified heuristics will be widely applied in managerial decision making.

We hope that more OR professionals will switch their attention to heuristics. Simple, understandable and usable heuristic approaches for solving managerial decision making problems are needed. Such approaches will undoubtedly be built around computer-based decision support systems. The growing number of well-designed DSS systems in industry may promote the use of quantitative heuristic decision techniques. Heuristics will be more attractive to managers, in the future, with the development of IT. Computer simulation may then be regarded as the last resort. Despite this, it is surprising how often such an approach is needed.

There are certain advantages in employing a simulation approach in management science and it may be the only way of tackling some managerial decision making problems.

List of References

Amrani, M. and Rand, G.,(1990). An eclectic algorithm for inventory replenishment for items with increasing linear trend in demand. Engineering Cost and Production Economics. 19: 261-266.

Anderson, D.R., D.J.Sweeney, and T.A. Williams (1991) *An Introduction to Management Science*, West Publishing Company.

Bartels, R.H. (1971) A stabilization of the simplex method, Numerische Mathematik 16, pp.414-434.

Beale, E.M.L. (1984) Mathematical programming, *Developments in operational research*, edited by R.W. Eglese and G.K. Rand, Pergarmon Press.

Beier, U. (1973) Zur Anwendung heuristischer Entscheidungsmethoden bei der Bestimmung eines Konsumprogramms. Zeitschrift für Betriebswirtschaft 43, 199-244.

Benders, J.F. (1962) Partitioning for solving mixed variables programming problems, Numerische Mathematik 4, pp.238-252.

Churchman, C.W. (1970) Operations research as a profession, Management Science, 17, B37-B53.

Dantzig, G.B. and P. Wolfe (1960) The decomposition principle for linear programs, Operations Research 8, pp.101-111.

Donaldson, W. A., (1977). Inventory replenishment policy for a linear trend in demand-an analytical solution. Op. Res. Quarterly 28: 663-670.

Fisher, M. L. (1981) The Lagrangean relaxation method for solving integer programming problems, Management Science 27, pp.1-18.

Forrest, J.J.H. and J.A. Tomlin (1977) Updating triangular factors of the basis to maintain sparsity in the product form simplex method, Mathematical Programming 13, pp.272-279.

Foulds, L. R. (1983) The Heuristic Problem-Solving Approach, J. Opl Res. Soc. Vol. 34, No. 10,pp.927-934.

Geoffrion, A.M. (1974) Lagrangian relaxation for integer programming, Mathematical Programming Study 2 pp.82-113.

Goldfarb, D. (1977) On the Bartels-Golub decomposition for linear programming bases, Mathematical programming 13, pp.272-279.

Goyal, S.K. and Gommes, L.F., (1982). Determination of replenishment intervals for a linear trend in demand. Working paper, Department of Quantitative Methods, Concordia University, Canada.

Herroelen, W. S. (1972) Heuristic Programming in Operations Mangement. Die Unternehmung 26, 213-231.

Himmelblau, D.M. (1979) *Decomposition methods*, Operations Research Support Methodology, edited by Albert G. Holzman, pp.483-514.

Hinkle, C.L., and A. A. Kuehn. (1970)Heuristc Modells: Mapping the Maze for Management. In: Information for Decision Making. Quantitative and Behavioral Dimensions. Hrsg. Rappaport, A.,Englewood Cliffs, S. 78-89.

Ho, J.K. (1987) Recent advances in the decomposition approach to linear programming, Mathematical Programming Study 31, pp.119-128.

Ho, J.K. and E. Loute (1981) An advanced implementation of the Dantzig-Wolfe decomposition algorithm for linear programming, Mathematical Programming 20, pp.303-326.

Ho, J.K., T.C. Lee and R.P. Sundarraj (1988) Decomposition of Linear Programs Using Parallel Computation, Mathematical Programming 42, pp.391-405.

Holmberg, K. (1992) Linear mean value cross decomposition: a generalization of the Kornai-Liptak Method. European Journal of Operational Research (EJO), **62(1)** pp. 55-73.

Isenberg, D.J. (1984) How senior managers think. Harvard Business Review, Nov-Dec, pp. 81-90.

Janczyk, W.K., and J.E. Beasly (1988) "Multiple-Model OR Packages, " Journal of Operational Research Society, Vol. 39, No. 5, pp.487-509.

Kaliski, J.A. and Y. Ye (1993) A short-cut potential reduction algorithm for linear programming. Management Science (MCI) **39(6)** pp. 757-776.

Kirsch, W. (1971) Entscheidungsprozesse. 2. Band: Informationsverarbeitungstheorie des Entscheidungsverhaltens. Wiesbaden pp.157.

Klein, H. K. (1971) Heuristische Entscheidungsmodelle. Neue Techniken des Programmierens und Entscheidens für das Management. Wiesbaden.

Lee, C.Y. (1991) An optimal algorithm for the multiproduct capacitated facility location problem with a choice of facility type, Computers Ops. Res. 18, pp.167-182.

Liewellyn, J., and R. Sharda (1990) "Linear Programming Software: 1990 Survey," OR/MS Today, Vol. 17, No.5,pp. 35-47.

Markowitz, H.M. (1957) The elimination form of the inverse and its applications to linear programming, Management Science 3, pp.255-269.

McMillan, C. (1970) Mathematical Programming: An Introduction to the Design and Application of Optimal Decision Machines. New York u. a.

Minsky, M. L. (1967) Künstliche Intelligenz. In: Information. Computer und künstliche Intelligenz. S. 191-208. Frankfurt-Main.

Moore, J.J. and S.J. Wright (1993), Optimization software guide.The Society for Industrial and Applied Mathematics.

Müller-Merbach, H. (1971)Operations Research. Methoden und Modelle der Optimalplanung. 2. Aufl., München 1971.
- (1973a) Heuristische Verfahren. In: Management-Enzyklopädie. Ergä nzungsband.S.346-355.
- (1973b) OR-Ansätze zur optimalen Abteilungsgliederung in Institution. In: Unternehmensführungund Organisation.Hrsg. Kirsch, S. 93-124.

Newell, A. (1969) Heuristic Programming: Ill-Structured Problems. In: Progress in Operations Research. Vol.III: Relationship between Operations Research and the Computer. Hrsg. Aronofsky,J.S.:New York U.a. S. 361-414.

Noel, M-C. and Y. Smeers (1987) Nested decomposition of multistage non-linear programs with recourse, Mathematical Programming 37, pp.131-152.

Perold, A.F. (1980) A degeneracy exploiting LU factorization for the simplex method, Mathematical Programming 19,

Pidd, M. (1988) *Computer Simulation in Management Science*, 2nd edition. John Wiley & Sons Ltd.

Pugh, D.S.; D.J. Hickson and C.R. Hinings (1979) *Writers on Organizations*, Penguin Books Ltd.

Reid, J.K. (1982) A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases, Mathematical Programming 24, pp.55-69.

Ritchie, E., (1984). The EOQ for linear increasing demand: A simple optimal solution. J.Oper.Res.Soc.,35:949-952.

Roy, T.J.V. (1983) Cross decomposition for mixed integer programming, Mathematical Programming 25, pp.46-63.

Roy, T. J. V. (1986) A cross decomposition algorithm for capacitated facility location, Operations Research 34, pp.145-163.

Silver, E. A. and Meal, H. C., (1973). A heuristic for selecting lot size quantities for the case of a deterministic time-varying demand rate and discrete opportunities for replenishment. Prod. Inv. Manage., 14(2): 64-74.

Silver, E. A., (1979). A simple inventory replenishment rule for a linear trend in demand. J.Oper.Res.Soc.,30:71-75.

Silver, E. A., R. Victor V. Vidal, and Dominique de Werra (1980) Atutorial on heuristic methods, Eur. J. Operational Research 5:153-162.

Streim, H. (1975) Heuristiche Lösungsverfahren Versuch einer Begriffsklärung, Zeitschrift für Operations Research, Band 19, Seite 143-162. Physics-Verlag, Würzburg.

Thesen, A. and Travis, L.E. (1992) *Simulation for Decision Making*, ISBN 0-314-83549-0; West Publishing Company.

Tonge, F. M. (1961) The Use of Heuristic Programming in Management Science. Management science 7, 231-237.

Tsado, A. (1985). Evaluation of the performance of lot-sizing techniques on deterministic and stochastic demand. Unpublished Ph.D. Thesis, University of Lancaster, UK.

Wheeling, R. F. (1969)Heuristic Search: Structured Problems. In: Progress in Operations Research, Vol. III: Relationship between Operations Research and the computer. Hrsg. Aronofsky,J. S, New York u.a. S.317-359.

Wiest, J. D. (1966) Heuristic Programs for Decision Making. Harvard Business Review 44, Heft 5, 129-143.

Yang, J., and G. K. Rand. (1993) An analytic eclectic heuristic for replenishment with linear increasing demand, International Journal of Production Economics, 32 (1993) 261-266.

Yurkiewicz,J., (1988) "Educational Operational Research Software: A Review," Interfaces, Vol. 18, No. 4, pp.59-71.

Appendices

1. **Available Commercial Optimisation Software**
2. **A Program for Testing Twelve Problems in the Case Study**

Appendix 1. Optimisation Software

1. CPLEX

Areas covered by the software

Ready - to run applications:

CPLEX Linear Optimise

CPLEX Mixed-integer Optimise

Optimisation libraries callable from C, Fortran, and Pascal programs:

CPLEX Callable Library

CPLEX Mixed-Integer Library

The CPLEX Linear Optimise and Callable Library solve linear programming problems. The CPLEX Mixed-Integer Optimise and Mixed-Integer Library solve integer programming problems as well as linear programming problems. Both linear and integer packages also solve network-structured problems with unlimited side constraints.

CPLEX products are designed to solve large, difficult problems where other linear programming solves fail or are unacceptably slow. CPLEX algorithms are exceptionally fast and robust, providing exceptional reliability, even for poorly scaled or numerically difficult problems.

Typical areas of application include large models in refining, manufacturing, banking, finance, transportation, timber, defence, energy, and logistics. CPLEX is also used heavily in academic research in universities throughout the world.

Additional comments

The CPLEX linear programming packages use a "modified primal and dual simplex" algorithm with multiple algorithm options for pricing and factorisation.

CPLEX solves are available in two forms:

The CPLEX Linear Optimise and Mixed-Integer Optimise are complete applications designed for ease of use. Because a complete on-line help system exists, most users never fully opening the package.

The CPLEX Callable Library and Mixed-Integer Library are in the form of callable routines that can be used to embed optimisation functionality within user-written applications. The callable products were designed to simplify development while providing the flexibility developers require. CPLEX reads linear and integer problems in several formats, including MPS format and CPLEX LP format. CPLEX also interfaces with several modelling languages, including GAMS, AMPL, and MPL.

To support academic research, CPLEX is offered at significant discounts to academic institutions.

2. C-WHIZ

Areas covered by the software

Linear programming models

Additional comments

C-WHIZ is an in-core implementation of the simplex algorithm. C-WHIZ accepts matrix input in standard MPS format or from the MPSIII database. C-WHIZ is undergoing continual enhancement. A version that removes the 32,000 row limit is being prepared.

3. LAMPS

Areas covered by the software

Linear programming and mixed-integer programming

Additional comments

LAMPS (Linear and Mathematical Programming System) offers a primal and a dual simplex algorithm for the solution of linear programs, and a branch-and-bound algorithm for mixed-integer programs. LAMPS is designed for the solution of large problems, although it will operate efficiently on small-sized and medium-sized problems. Most standard input formats are acceptable, and output (solution) reporting is very flexible.

Mixed-integer problems may define data in terms of S1 or S2 sets, general integer, binary, and semicontinuous variables.

Algorithms of LAMPS are also avaible for direct use with MAGIC (a matrix generation and reporting system) and GAMS.

4. LINDO

Areas covered by the software

Linear programming, mixed-integer linear programming, quadratic programming.

Additional comments

LINDO uses simplex and active set algorithms for linear and quadratic programming, and a branch-and-bound approach for mixed-integer programming.

5. MINOS

Areas covered by the software

Linear programming, unconstrained and constrained non-linear optimisation.

Additional comments

Input to MINOS 5.4 is via MPS files (which contain information for the linear parts of the objective function and constraints), SPECS files (which specify the problem types and set various parameters), and Fortran codes (which calculate objective and constraint functions, if non-linear). The GAMS system can be used as an alternative user interface. See the entry on GAMS for details.

6. OSL

Areas covered by the software

Linear programming, convex quadratic programming, and mixed-integer programming problems.

Additional comments

For linear programming, primal and dual versions of the simplex method are implemented. a branch-and-bound technique is used for mixed-integer programming.

Volume 31 (1992) of the IBM *Systems Journal* contains eight articles related to OSL. An overview of the product may be found in the article cited below.

7. BQPD

Areas covered by the software

BQPD solves quadratic programming problems. A general form of the problem is solved that allows upper and lower bounds on all variables and constraints. If the Hessian matrix Q is positive definite, then a global solution is found. The method can also be used when Q is indefinite, in which case a Kuhn-Tucker point that is usually a local solution is found.

Additional comments

The code implements a null-space active set method with a technique for resolving degeneracy that guarantees that cycling does not occur even when roundoff errors are present. Special features include a constraint prescaling routine and full documentation through comments in the code. Special arrangements can be made for use in a commercial environment.

8. LSSOL

Areas covered by the software

LSSOL is a Fortran package for linearly constrained linear least squares problems and convex quadratic programming. LSSOL is designed to solve a class of linear and quadratic programming problems of the following general form:

$$\text{minimise } \{f(x) : x \in \Re^n\}$$
$$\text{subject to } l \le \begin{Bmatrix} x \\ Cx \end{Bmatrix} \le u,$$

where C is an m (L × N) matrix (m L may be zero) and f(x) is one of the following:

FP:	None	(find a feasible point for the constraints)
LP:	c'x	(a linear program)
QP1:	½ x' Ax	A symmetric and positive semidefinite,
QP2:	c'x + ½ x' Ax	A symmetric and positive semidefinite,
QP3:	½ x' A' Ax	A m × n upper trapezoidal,
QP4:	c'x + ½ x' A' Ax	A m × n upper trapezoidal,

LS1:	½ | b-Ax | ²	A m × n,
LS2:	c'x + ½ | b-Ax | ²	A m × n,
LS3:	½ | b-Ax | ²	A m × n upper trapezoidal,
LS4:	c'x + ½ | b-Ax | ²	A m × n upper trapezoidal,

with c an n-vector and b an m-vector.

Additional comments

LSSOL is essentially identical to the routine E04NCF of the NAG Fortran Library. E04NCF was introduced at Mark 13. LSSOl was first distributed by the Office of Technology Licensing at Stanford in 1986. Since that time, the routine has been continually revised. Users with older versions of LSSOL should consider obtaining a copy of the most recent version.

9. QPOPT

Areas covered by the software

QPOPT is a FORTRAN package designed to solve linear and quadratic programming problems of the following general form:

$$\text{minimise } \{f(x) : x \in \Re^n\}$$
$$\text{subject to } l \le \begin{Bmatrix} x \\ Ax \end{Bmatrix} \le u,$$

where A is an m (L× N) matrix (mL may be zero) and f(x) is one of the following:

FP:	None	(find a feasible point for the constraints)
LP:	c'x	(a linear program)
QP1:	½ x' Hx	H symmetric,
QP2:	c'x + ½ x' Hx	H symmetric,
QP3:	½ x' H' Hx	H m × n upper trapezoidal,
QP4:	c'x + ½ x' H' Hx	H m × n upper trapezoidal,

with c an n-vector. In QP1 and QP2, there is no restriction on H apart from symmetry. If the quadratic function is convex, a global minimum is found;

otherwise, a local minimum is found. The method used is most efficient when many constraints or bounds are active at the solution. If H is positive semidefinite, it is usually more efficient to use the package LSSOL.

Additional comments

QPOPT is essentially identical to the routine E04NCF of the NAG FORTRAN Library. E04NCF was introduced at Mark 16.

The method of QPOPT is similar to the method of QPSOL, which was distributed by Stanford University between 1983 and 1991. However, QPOPT is a substantial improvement over QPSOL in both functionality and reliability.

10. OB1 (Optimisation with Barriers-1)

Areas covered by the software

Linear programming

Additional comments

The software has been in development since 1987. Linear programs with up to 40,000 constraints and 180,000 variables have been solved. The main algorithm is the primal-dual interior-point method, with Mehrotra's predictor-corrector strategy.

11. BTN

Areas covered by the software

Unconstrained minimisation in a parallel computing environment. The software is especially suited to problems with a large number of variables.

Additional Comments

BTN uses a block, truncated Newton method based on a line search.

12. GAUSS

Areas covered by the software

The software consists of nine packages that are designed for use with the GAUSS matrix programming language. Four of these packages relate to optimisation; the other five relate to statistical analysis. The four relevant packages are

OPTIMUM	Unconstrained optimisation
MAXLIK	Maximum likelihood estimation
NLSYS	Solving non-linear equations
SIMPLEX	Linear programming

Additional comments

These packages are being revised and expanded. Four methods can be chosen: Newton's method, quasi Newton Methods, steepest-decent, and Polak-ribiere conjugate gradient method.

13. IMSL

Areas covered by the software

Figures 1 and 2 provide a quick reference to the optimisation routines of the FORTRAN Library. Only part of this coverage is available for the C Library. There are also routines for solving systems of non-linear equations .

Additional comments

Users can include the software in their derivative works with a licensing agreement. Contact IMSL,Inc., Sales, for more details.

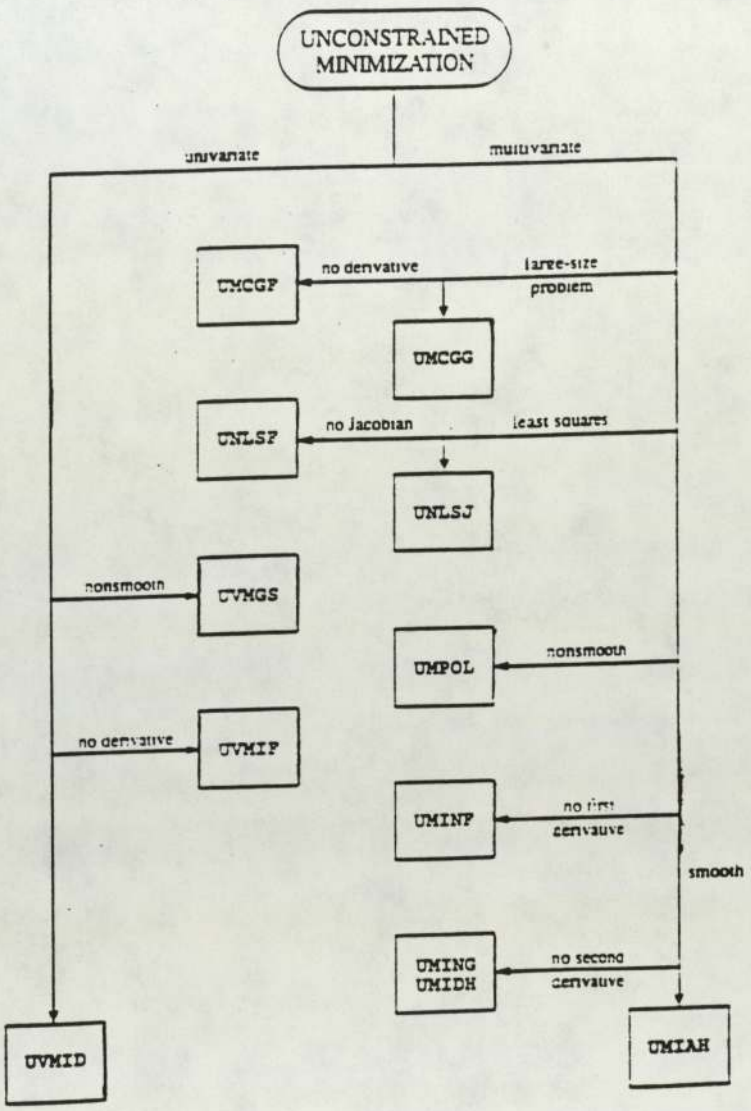

FIG. 1. *Unconstrained minimization subroutines in the IMSL library.*

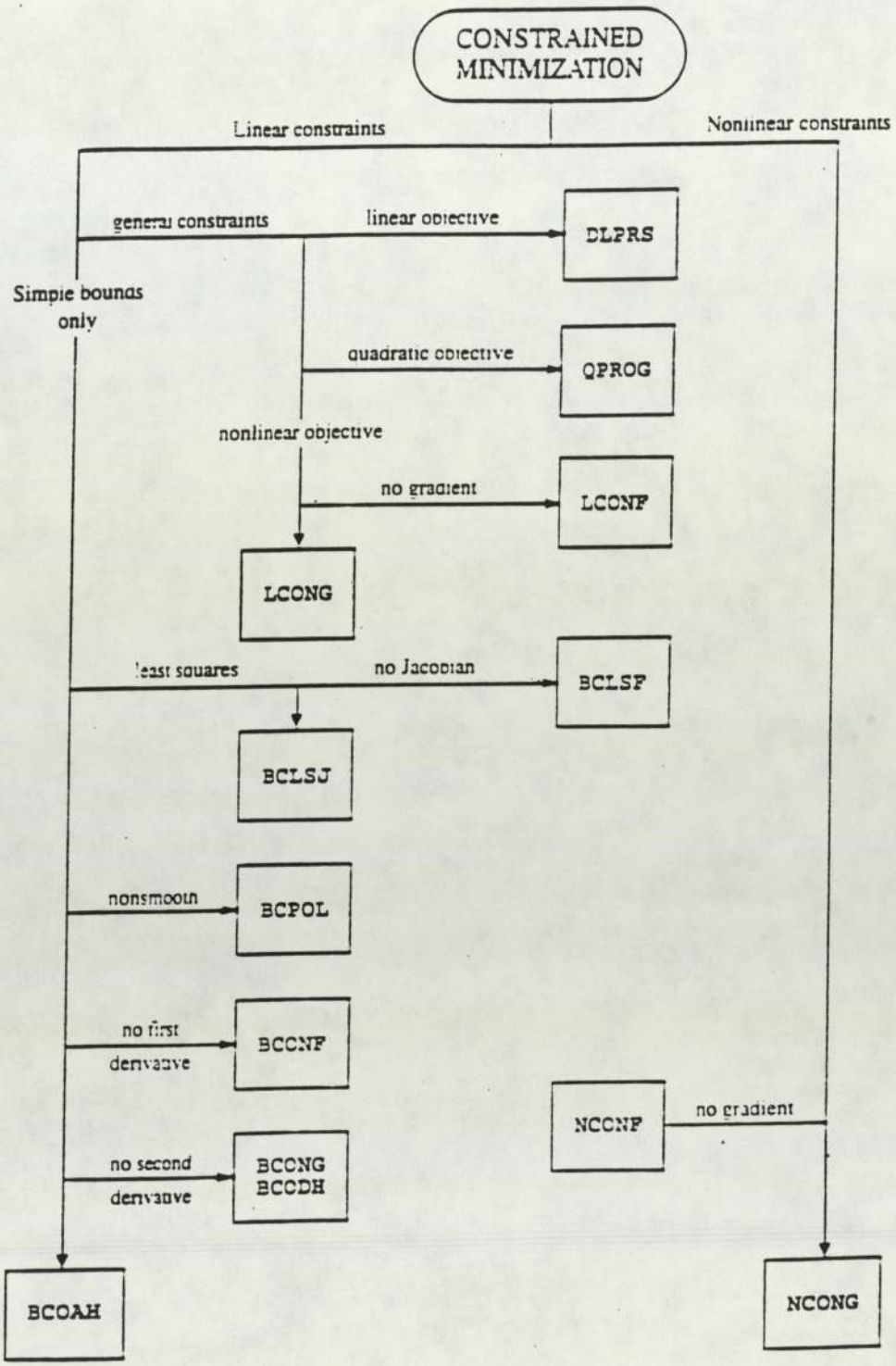

FIG. 2. *Constrained minimization subroutines in the IMSL library.*

14. LANCELOT

Areas covered by the software

Unconstrained optimisation problems, systems of non-linear equations, non-linear least squares, bound-constrained optimisation problems, and general non linearly constrained optimisation problems. Special emphasis is placed on large-scale problems.

Additional comments

The LANCELOT package uses an augmented Lagrangian approach to handle all constraints other than simple bounds.

15a. NAG (C Library)

Areas covered by the software

It covers linear programming, quadratic programming, minimisation of a non-linear function(unconstrained or bound constrained), minimisation of a sum of squares.

15b. NAG (Fortran Library)

Areas covered by the software

It covers linear programming; mixed-integer linear programming; quadratic programming; minimisation of a non-linear function (unconstrained, bound constrained, linearly constrained, and non linearly constrained); and minimisation of a sum of squares (unconstrained, bound constrained, linearly constrained, and non linearly constrained).

Additional comments

For problem with non-linear constraints, a sequential QP algorithm is used.

The NAG Fortran Library is updated to a new Mark from time to time. Mark 16, which contains improved routines for linear and quadratic programming was released in the first half of 1993. Further developments in optimisation routines are planned for future Mark versions.

16. OPTIMA

Areas covered by the software

Unconstrained optimisation, constrained optimisation, sensitivity analysis. Software is written in Fortran 77.

17. PORT3

Areas covered by the software

General minimisation, non-linear least squares, separable non-linear least squares, linear inequalities, linear programming, and quadratic programming. The non-linear optimises have unconstrained and bound-constrained variants.

Additional comments

The non-linear optimises use trust-region algorithms. Software is written in ANSI Fortran 77. The general minimisation routines use either a quasi-Newton approximation to the Hessian matrix or a Hessian provided by the caller.

18. PROC NLP

Areas covered by the software

Non-linear minimisation or maximisation with linear constraints.

Additional comments

PROC NLP is part of the SAS/OR (Operations Research) package. The current version of PROC NLP is experimental. When PROC NLP goes into production, various extensions will be implemented. In particular, a special algorithm for optimising a quadratic function with linear constraints will be offered, non-linear constraints will be implemented by an augmented Lagrangian approach, and a reduced-gradient version with sparse LU decomposition will be provided for large and sparse systems of linear constraints.

19. TENMIN

Areas covered by the software

Unconstrained optimisation.

Additional Comments

The package allows the user to choose between a tensor method for unconstrained optimisation and a standard method based on quadratic model.

20. TN

Areas covered by the software

Unconstrained minimisation and minimisation subject to bound constraints. The software is especially well suited to problems with large numbers of variables.

21. TNPACK

Areas covered by the software

Non-linear unconstrained minimisation of large-scale separable problems. A truncated Newton method for unconstrained minimisation has been specifically developed for large-scale separable problems.

22. UNCMIN

Areas covered by the software

Unconstrained optimisation.

Additional comments

UNCMIN is a modular package based on a Newton or quasi-Newton approach. It allows the user to select from various options for calculating or approximating derivatives and for the step-selection strategy.

23. VE08

Areas covered by the software

Bound-constrained non-linear optimisation with an emphasis on large-scale problems.

Additional Comments

VE08 is a line-search method with a search direction obtained by a truncated conjugate gradient technique.

2. Programs for Testing Twelve Problems in the Case Study

a. PROGRAM Donaldson(input,output);

```
CONST
        AssumedZero = 0.00000001;

TYPE
        Vector = array[1..500] of real;

VAR
        Z,G,T,Y,X,Z1,Z2,TT: Vector ;
        i,j,k,l,n,n1,n2,kk : integer ;
        a,b,H,c1,c2 : real ;      {The problem parameters}
        Test,Cost,m,prod,cost1,cost2,zmin,zmax,ztest:real;

FUNCTION Calcul(r:real;nn:integer):real ;
VAR
        P:real;
begin
        X[1]:=r;
        For l:=2 to nn do
                X[l]:=sqrt(3-2/X[l-1]);
                P:=X[l];
                for l:=2 to nn do
                        P:=P*X[l];
        Calcul :=1+H*b/a-P;
end;

PROCEDURE Resolve(a,b,H,c1,c2:real);
begin
        Test := c1/(H*H*H*b*c2);
        Z[2]:=sqrt(3); Y[1]:=1; G[1]:=0;
        Y[2]:=Z[2]; G[2]:=1/(3*sqrt(3));
        i:=2;
        repeat
                i := i+1;
                Z[i]:=sqrt(3-(2/Z[i-1]));
                Y[i]:=Y[i-1]*Z[i];
                G[i]:=(1/(Y[i]*Y[i]*Y[i]));
                for k:=2 to i-1 do
                G[i]:=G[i]+(1/(Y[i]*Y[i]*Y[i]))*Y[k]*Y[k]*Y[k]*(1-(1/z[k]));
        until G[i]-G[i-1] <= Test ;

        for j:=1 to i-1 do
                T[j]:=Y[j]/Y[i-1]*H;
end;

PROCEDURE Timing(nn:integer); {Procedure determining the optimal
                                replenishmnt points for a number
```

```
                                        nn of replenishments}
begin
        for k:=1 to nn do
        begin
                Prod:=1;
                for kk:=1 to k do
                Prod:=Prod*x[kk];
                TT[k]:=a/b*(Prod-1);
        end;
end;

PROCEDURE Find(nn:INTEGER);{Procedure finding the optimal
                                        number of replenishments nn}
begin
        Zmin:=1;
        Zmax:=100;
        Ztest:=50;
        repeat
                if calcul(Ztest,nn)>0 then
                        begin
                                ZMIN :=ZTEST;
                                ZTEST:=(ZMAX+ZTEST)/2;
                        end
                else
                begin
                        ZMAX:=ZTEST;
                        ZTEST:=(ZMIN+ZTEST)/2;
                end;
        until abs(calcul(ZTEST,nn))< AssumedZero;
end;

PROCEDURE Display(ii:integer;zz:vector;cc:real);
{Procedure displaying results}

begin
        writeln('a= ',a);
        writeln('b= ',b);
        writeln('H= ',H);
        writeln('c1=',c1);
        writeln('c2=',c2);
        writeln;writeln;
        writeln(' OPTIMAL POLICY');
        writeln;
        writeln('    ',ii,' Replenishments');
        writeln;
        writeln('      t(0)= 0');
        for k :=1 to ii do
        writeln('    t(',k,')=',zz[k]);
        writeln('  COST IS ',CC);
end;
```

```
begin  {Main program starts here}
       writeln('ENTER a,b,H,c1,c2 ');
       readln(a); readln (b); readln (H); readln (c1); readln (c2);
       m:=c1/c2;
       IF a=0 THEN
              begin
                     Resolve(a,b,H,c1,c2);
                     n:=i-1;
                     Cost:=n*m+a*H*H/2+b*H*H*H/3;
                     FOR k:=1 TO n-1 DO
                     Cost:=Cost-T[k]*(T[k+1]-T[k])*(a+b/2*(T[k+1]+T[k]));
                     Display(n,T,Cost);
              end
       else
       begin
              Resolve(0,b,H+a/b,c1,c2);
              k:=0;
              repeat k:=k+1
              until T[k]>a/b;
              n1:=i-k-1;
              n2:=i-k;
              Find(n1);Timing(n1);
              for j:=1 to n1-1 do
              Z1[j]:=TT[j];
              Z1[n1] := H;
              Cost1:=n1*m+a*H*H/2+b*H*H*H/3;
              for k:=1 to n1-1 do
                     Cost1:=Cost1-Z1[k]*(Z1[k+l]-Z1[k])*(a+b/2*(Z1[k+1]+Z1[k]));
              Find(n2);Timing(n2);
              for j:=1 to n2-1 do
                     Z2[j]:=tt[j];
              Z2[n2]:=h;
              Cost2:=n2*m+a*H*H/2+b*H*H*H/3;
              for k:=1 to n2-1 do
                     Cost2:=Cost2-Z2[k]*(Z2[k+1]-Z2[k])*(a+b/2*(Z2[k+1]+Z2[k]));
              if Cost1<Cost2 then Display(n1,Z1,Cost1)
              else Display(n2,Z2,Cost2);
       end;
end.
```

b. Program AnalyticEclectic (input, output);

```
USES CRT;
CONST
        LargeValue = 10000000;
TYPE
        rmat1 = array[1..365,1..4] of real;
        rvec1 = array[1..365] of real;
VAR
        t0, tn, c0, c1, d0, d1: real;
        H,lemda,t: real;
        a,b,c,d,Intv: real;
        j,k,l,choice1,choice2: integer;
        RN,optRN:  integer;
        RTime,RIntv,RQ,RC: rvec1;
        optTC,optLemda: real;
        optRTime,optRQ,optRC: rvec1;
        TC,TQ: real;

        outfile: text;

Procedure Initialisation; {Input: data}
begin
        ClrScr;
        assign(outfile,'a:re-ling.dat');
        rewrite(outfile);
        begin {Remove Brakets to Choose a Problem}
        {   d0 := 0;  d1 := 900;   t0 := 0;  tn := 1;   c0 := 9;  c1 := 2;}
        {   d0 := 0;  d1 := 900;   t0 := 0;  tn := 2;   c0 := 9;  c1 := 2;}
        {   d0 := 0;  d1 := 100;   t0 := 0;  tn := 4;   c0 := 100;  c1 := 2;}
        {   d0 := 0;  d1 := 1600;  t0 := 0;  tn := 3;   c0 := 42;  c1 := 0.56;}
        {   d0 := 6;  d1 := 1;   t0 := 0;  tn := 11;    c0 := 30;  c1 := 1;}
        {   d0 := 6;  d1 := 1;   t0 := 0;  tn := 11;    c0 := 50;  c1 := 1;}
        {   d0 := 6;  d1 := 1;   t0 := 0;  tn := 11;    c0 := 60;  c1 := 1;}
        {   d0 := 6;  d1 := 1;   t0 := 0;  tn := 11;    c0 := 70;  c1 := 1;}
        {   d0 := 6;  d1 := 1;   t0 := 0;  tn := 11;    c0 := 90;  c1 := 1;}
        {   d0 := 100;  d1 := 150;  t0 := 0;  tn := 1;   c0 := 30;  c1 := 2;}
        {   d0 := 100;  d1 := 150;  t0 := 0;  tn := 1.5;   c0 := 30;  c1 := 2;}
          d0 := 100;  d1 := 150;  t0 := 0;  tn := 2;   c0 := 30;  c1 := 2;
              writeln(outfile,d0,d1,tn-t0,c0,c1);
        end;
        H := tn-t0;
end;

Procedure Unification(t,lemda: real);
begin
        a := lemda*d1;
        b := (d0+d1*t)/2;
        c := 0;
        d := -c0/c1;
end;
```

```
Procedure Confirmation;
begin
        ClrScr;
        writeln('May I discribe your replenishment problem as follows:');
        writeln('Initial time: ',t0:10:4);
        writeln('Ending time: ',tn: 10:4);
        writeln('Constant demand: ',d0: 10:4);
        writeln('First demand rate: ',d1: 10:4);
        writeln('Fixed replenishmnet cost: ',c0: 10:4);
        writeln Unit inventory cost: ',c1: 10:4);
end;

procedure PatternChoice;
begin
        writeln('Which heuristic would you like to choose?');
        writeln;
        writeln('1  Silver');
        writeln('2  CLUC');
        writeln('3  CPPA');
        writeln('4  Ritchie');
        writeln('5  Eclectic: Amrani-Rand');
        writeln('6  Generalized Eclectic: Yang-Rand');
        writeln('7  Simulation: Lemda > 0');
        writeln;
        readln(choice1);

        if choice1 = 1 then lemda := 2/3
        else if choice1 = 2 then lemda := 1/6
        else if choice1 = 3 then lemda := 1/3
        else if choice1 = 4 then lemda := 0.43
        else if choice1 = 7 then
        begin
                writeln('Please enter your Lemda = ');
                readln(lemda);
        end;
end;


Procedure OptPatternChoice;
begin
        if choice1 = 6 then
        begin
                k := 1; optTC := Largevalue;
                repeat
                        begin
                                lemda := k/100;
                                ReplenishmentPlan(lemda);
                                MultiIndex(RIntv,RTime,RN);
                                if (TC < optTC) then
```

```
                        begin
                                optTC := TC;
                                optLemda := lemda;
                                optRN := RN;
                        end;
                        k := k+1;
                end;
        until k = 100;
        lemda := optlemda;
    end;
end;

Function POWER(X,Y : real) : real;
begin
    if Abs(X) < 0.000000000001 then Power :=  0;
    if X > 0 then Power := EXP(Y*LN(X));
    if X < 0 then Power := -EXP(Y*LN(Abs(X)));
end;

Function MAX(X1,X2: real) : real;
begin
    if (X1 > X2) then   Max := X1
    else          Max := X2;
end;

Procedure CubicRootFinding(a,b,c,d: real);
var
    p,q,r: real;
    temp,delta,tempt,temp1,temp2,temp3: real;
Begin
    begin
        Intv := 0;
        b := b/a;
        c := c/a;
        d := d/a;
        p := -b*b/3+c;
        q := 2*b*b*b/27-c*b/3+d;
        delta := q*q/4+p*p*p/27;
    end;
    {output: intv - the largest real root of a cubic equation}
    begin
        if (delta = 0) and (p = q) then Intv := -b/3; {This is unique root}
        if  delta < 0 then {Calculate the three roots}
        begin
            r := Sqrt(-p*p*p/27);
            temp := -q/(2*r);
            temp := temp/Sqrt(1-temp*temp);
            temp := Pi/2 - ArcTan(temp);
            temp1 := 2*SQRT(-p/3)*COS(temp/3);
            temp2 := 2*SQRT(-p/3)*COS(temp/3+2*PI/3);
```

```
                temp3 := 2*SQRT(-p/3)*COS(temp/3+4*PI/3);
                Intv := Max(temp1,temp2);
                Intv := Max(Intv,temp3);
                Intv := Intv-b/3;
            end;
            {To choose the right root}
            if delta > 0 then        { <=> discrimant > 0 }
            begin
                temp1 := -q/2+Sqrt(delta);
                temp2 := -q/2-Sqrt(delta);
                temp1 := Power(temp1,1/3);
                temp2 := Power(temp2,1/3);
                Intv := temp1+temp2-b/3;
            end;
        end;
        if intv < 0 then writeln('Funny! Please check your lemda and data!!');
end;

Procedure ReplenishmentPlan(lemda:real);
var
        t: real;
begin
        t := t0;
        j := 1;
        RTime[1] := t0;
        repeat
            Unification(t,lemda);
            CubicRootFinding(a,b,c,d);
            RIntv[j] := Intv;
            t := t+Intv;
            j := j+1;
            RTime[j] := t;
        until t >= tn;
        RN := j-1;
        RTime[RN+1] := tn;
        RIntv[RN] := tn-RTime[RN];
end;

Procedure MultiIndex(RIntv,RTime: rvec1; RN: integer);
var
        i: integer;
begin
        TQ := 0; TC := 0;
        i :=1;
        repeat
            RQ[i] := d0*RIntv[i]+(d1/2)*(Sqr(RTime[i+1])-Sqr(RTime[i]));
            RC[i] := ((1/2)*(d0+d1*RTime[i])+(d1/3)*RIntv[i])*Sqr(RIntv[i]);
            RC[i] := c0+c1*RC[i];
            TQ := TQ+RQ[i];
            TC := TC+RC[i];
```

```
                i := i+1;
        until i = RN+1;
end;

Procedure BeautfulFinishing;
var
        temp: real;
begin
        temp := tn-RTime[RN-2];
        temp := ((1/2)*(d0+d1*RTime[RN-2])+(d1/3)*temp)*Sqr(temp);
        temp := c0+c1*temp;
        if temp < RC[RN-2]+RC[RN-1] then
        begin
                writeln('beautful finishing next time, Sir.')
        end;
end;

Procedure Display(RC,RQ: rvec1);
begin
        if choice1 = 1 then writeln(outfile,'Silver:' );
        if choice1 = 2 then writeln(outfile,'Continuous Least Unit Cost:');
        if choice1 = 3 then writeln(outfile,'Continuous Part Period Agorithm:');
        if choice1 = 4 then writeln(outfile,'Ritchie Infinity:');
        if choice1 = 5 then writeln(outfile,'Amrani-Rand Eclectic:');
        if choice1 = 6 then writeln(outfile,'Yang-Rand Algorithm:');
        if choice1 = 7 then writeln(outfile,'Mr./Ms. X  Heuristics:');
        writeln(outfile,'lemda = ',lemda);
        writeln(outfile,'No.     Rep. Time        Rep. Quantity       Rep. Cost');
        j := 1;
        repeat
                writeln(outfile,j,'   ',RTime[j]: 6:4,'   ',RC[j]: 6:4,'   ',RQ[j]: 6:4);
                j := j+1;
        until j = RN+1;
        writeln(outfile,'Total cost = ',TC);
end;

BEGIN
        Initialisation;
        Confirmation;
        PatternChoice;
        if choice1 = 6 then
        begin
                k := 1; optTC := Largevalue;
                repeat
                        begin
                                lemda := 0.01*k;
                                ReplenishmentPlan(lemda);
                                MultiIndex(RIntv,RTime,RN);
                                writeln(k,' % ');
                                writeln(outfile,'   ',lemda: 6:4,'  ',TC: 6:4,'  ',RN);
```

```
                        if (TC < optTC) then
                        begin
                                optTC := TC;
                                optLemda := lemda;
                                optRN := RN;
                        end;
                        k := k+1;
                end;
        until k = 100;
        lemda := optlemda;
        writeln(outfile,'opt:   ',lemda: 6:4,'  ',optTC: 6:4,'  ',optRN);
end;
ReplenishmentPlan(lemda);
Multiindex(RIntv,RTime,RN);
{BeautfulFinishing; this is for ending}
Display(RC,RQ);
writeln('TC = ',TC);
Close(outfile);
End.
```