HUMAN RESOURCE ALLOCATION IN A CALL CENTRE

GILLES LE CORROLLER

Master of Science by Research in Pattern Analysis and Neural Networks

Supervisor: Dr Ian Nabney

THE UNIVERSITY OF ASTON IN BIRMINGHAM

September 1997

THE UNIVERSITY OF ASTON IN BIRMINGHAM

HUMAN RESOURCE ALLOCATION IN A CALL CENTRE

GILLES LE CORROLLER

Master of Science by Research in Pattern Analysis and Neural Networks, 1997

Thesis Summary

A novel approach to the problem of resource allocation in telephone call demand satisfaction has been proposed by Prof David Lowe and Dr Ian Nabney in a feasibility study. The method, inspired by neural network approaches to combinatorial optimisation, has the advantage of allowing a flexible use of a multi-skilled workforce, proposing about 25% fewer agents compared to figures produced by a traditional statistical method based on agents with single skills and separate queues.

The purpose of this thesis is to develop and investigate some of these assumptions. One of these assumptions is that there is an unlimited pool of people in each skills mix. As this may not be appropriate for more highly qualified profiles, a method to cap the number of agents is described and its effects are investigated. It has been implemented and added to the earlier algorithm.

To investigate the distribution of the calls across time, we developed a stochastic model of the queues based on the Erlang C model currently used in scheduling software. This model is more realistic than the assumption, made in the feasibility study, that the calls are uniformly distributed. In our new model, the arrivals are assumed to follow a Poisson process and the service time distribution is described by an hyper-exponential distribution, which takes into account the multi-skilled ability of each operator. This leads us to increase the number of agents required to achieve a proper service level, but nevertheless our approach still proposes 20% fewer agents compared to existing methods.

The reliability of our method is tested with different data sets. In addition, some simulations were carried out to validate our new approach. The results, which are quite encouraging, are presented in detail.

Another aspect of our approach was partially investigated. Instead of grouping people together in small numbers of groups with similar skills profiles, we grade each individual member of the workforce with his own unique skills profile. Consequently the incoming calls will be allocated to each individual and not to groups of agents. Some results and conclusions about this new issue are detailed.

Keywords: call centre, multi skilled agent, connectivities, stochastic modelling of the queues, simulation

Acknowledgements

I would like to thank Steve Grant at CallScan for his help and for his kindness during the several meetings we had.

I am also thankful to CallScan for the financial support provided by paying my tuition fees.

I finally thank Dr Ian Nabney for his help and his useful advises during this nine month project.

Contents

List of Figures

Chapter 1

Introduction

1.1 Presentation of the problem

The problem considered here is that of human resource allocation in a telephone call centre. The motivation behind the project is to devise a method which tackles this problem in a new way.

A typical telephone call centre deals with a set of incoming telephone calls. The calls correspond to different types of queries or services available for customers. Each customer has to dial a phone number which corresponds to the specific type of service he wants. Therefore each telephone number corresponds to a specific type of skill from the agent employed to answer the queries. According to the number dialled by the customer, the calls will be routed in one of the queues of the call centre. Then the customer will have to wait in the queue until his call is allocated to an agent qualified to cope with this type of call.

On the other side of a call centre, there are the agents, whose job is to answer the incoming calls. Each agent has a skills profile which is a measure of their ability at answering certain types of calls. The problem faced by the call centre management is to allocate the incoming calls to the agents in order to answer the calls effectively and to minimise the waiting time for each call. Typically the aim is to allocate sufficient numbers of agents of the required skills in order to provide a specified service level, for instance 95% of calls answered within 30 seconds. It is also advantageous to minimise the number of staff employed to answer the calls, while still maintaining a reasonable level of service.

All the methods of forecasting the number of agents needed to provide the right service level with the smallest number of staff rely on parameters which describe the daily activity of the call centre. The two main parameters used to describe the incoming calls in each queue are:

- The call volume which is the average number of incoming calls in a given span of time.
- The call duration which is the average duration of a call. This parameter is strongly related with the mean service rate provided by the agents.

With such parameters, it is possible to derive the density of telephone calls. Of course the call density varies throughout the day and also throughout a week in a characteristic manner, which may be specific to the business domain.

Once the different parameters are known, the forecasting method provides the optimum number of agents required in the call centre and the expected performance of the call centre. The most important performance measure given by a forecasting method is the service level, i.e. the percentage of calls answered within a period. A good service level is between 95% and 98% of incoming calls answered within 30 seconds. Another essential performance measure is the average delay of waiting for an incoming call before being answered. This measure is strongly linked with another one, the percentage of abandoned calls which will increase if the waiting time is too long. Another important performance measure is the cost of the required number of agents in each pool, for labour is the largest cost in a call centre.

The purpose of this project is to device and test a new approach to the resource allocation, so that the required number of staff will be smaller than the one obtained with the traditional approach. The two approaches are presented in the following sections.

1.2 Current method

In this section the forecasting method currently used in the call centre will be described and illustrated.

In the traditional way of dealing with incoming calls, the agents are gathered in as many pools as there are queues. All the calls in a queue will be answered by agents of the corresponding pool, who have the skills to respond to that specific queue. This philosophy is based on statistical independence assumptions whereby each queue of calls is modelled as an independent statistical process. This situation is depicted schematically in Figure 1.1. The traditional forecasting attempts to determine the optimum number of agents within each pool using the mean observed behaviour of the queues.

It is worth noting that the quality of the answer is not taken into account in this forecasting method, for the agents are assumed to have all the skills required for answering the phone calls that they have to deal with. Thus the standard of service will be measured by the service level and the number of abandoned calls.

The drawback with this approach is that no account can be made of correlations between the different queues. However, from the observations of the actual data, the call densities are clearly correlated. Another drawback is that no allowance is made for the fact that some multi-skilled agents could answer calls from several different queues. Such allowance would allow us to use up possible spare capacity and to allow fluctuations in call numbers to be spread more widely. The purpose of developing a new forecasting method is to tackle this drawbacks.

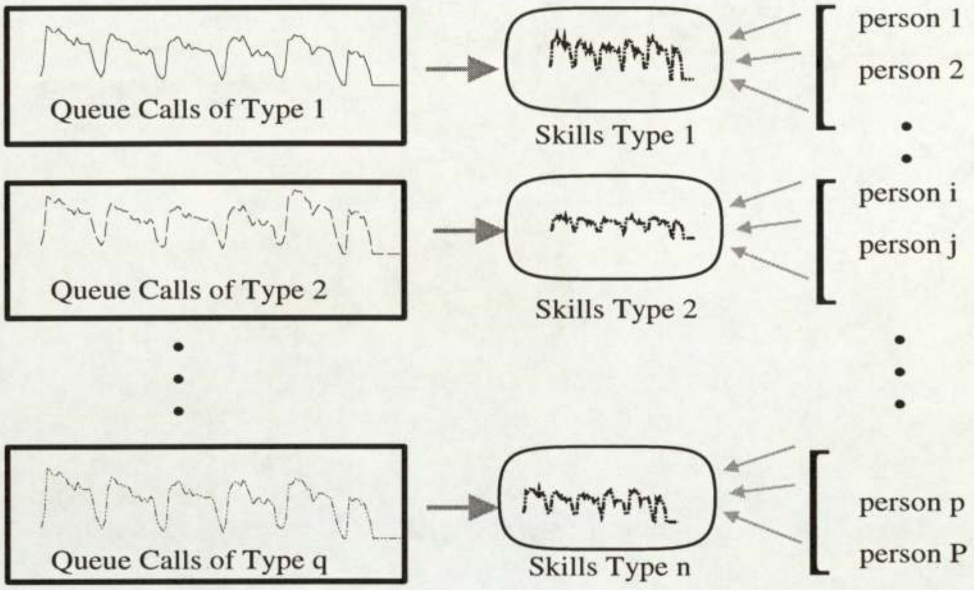


Figure 1.1: Schematic diagram of the traditional approach to allocate agents to just one specific queue, depending upon their skills type.

1.3 New approach

An alternative approach is to assume that the agent population can be pooled by tagging each agent with their ability at answering each of the possible queues. Hence in principle, any call could be answered by any agent, if no agents of highest ability were available. This situation is depicted in Figure 1.2.

The first step in this approach is to define the skills profile of each agent, which will describe how well they can answer each type of incoming call. Then an algorithm can be devised to forecast the optimum number of agents required to answer the calls.

The number of agents has to be big enough to provide a good service level and to give a sufficiently high quality of service. In fact, in this approach, we admit that the calls can be answered by agents not best qualified so that we can hope to decrease the number of agents needed but we have consequently to be aware of the quality of the answer. That's why this approach requires a measure of quality of service provided to the customer.

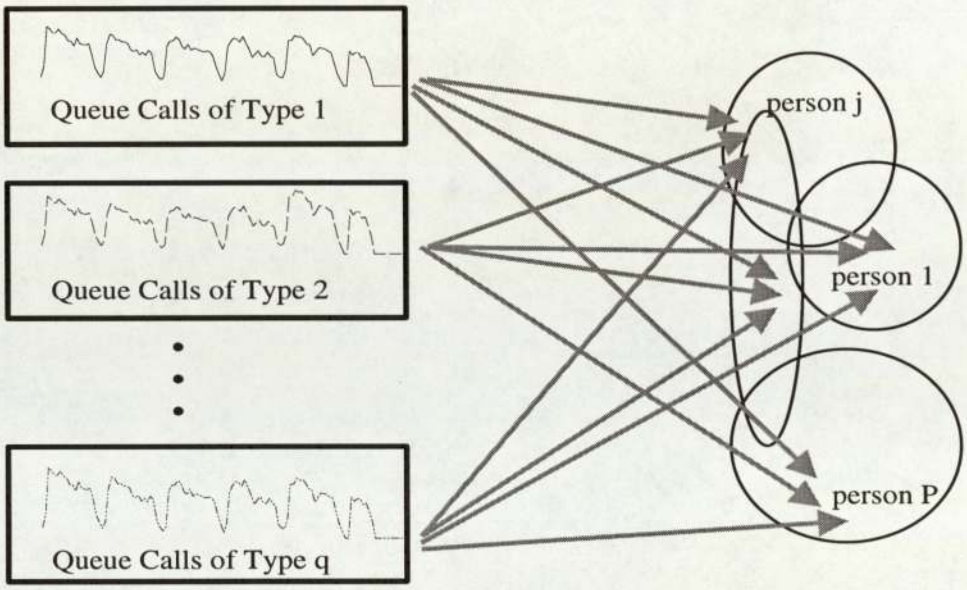


Figure 1.2: Schematic diagram of our novel approach: Each agent may respond to any of the queues with a capacity depending upon their overall skills profile.

1.4 System overview

In this section, we shall describe how this new scheduling method fits into the overall scheme of a call centre.

In our new approach, we redefine the way telephone calls are allocated to agents in a call centre. Obviously we expect this new method to require significantly less agents than the traditional one. However we will have to look closely at the quality of service provided to the customers, as it might be affected by our new approach. Thus our aim is to predict the number of agents required in our new approach, whilst producing some measures of the quality of service.

The first step in a forecasting method is to get some data on the daily activity of the call centre. The two main parameters describing this activity are the call volume and the average call duration, which are defined in the first section of this chapter. Using this two parameters, we can either compute the call density, i.e. the average number of calls active at an instant in the call centre, or derive the total number of agents required in the system using a stochastic modelling of the queues. Thus we would be able to forecast the required number of agents in the system.

Another important point is to measure the quality of service provided to the customers. The first aspect of it is the service level, i.e. the percentage of incoming calls answered within a given delay. The second one is to measure how well the skills of the agents correspond to the ones required by the different types of calls they are answering. This last measure is based on the skills profile of the agents. By being more or less demanding in our requirements of these two measures, we would be able to increase or decrease the number of agents in the system.

1.5 Thesis overview

Chapter 2 summarises the feasibility study [1] and its conclusions. The project follows on from this feasibility study and aims to investigate some of the assumptions made in the feasibility study.

Chapter 3 describes the methods tried to cap the number of agents in certain skills mix pools, the results obtained and the conclusions.

Chapter 4 presents the work done on the stochastic modelling of the queues. The basic elements of the queueing theory are explained and the stochastic model used to describe the queues of incoming calls is detailed.

Chapter 5 summarises the tests done in order to validate our approach. We have run our algorithm with different data sets and observed and compared the results. We have also written a simulation program in order to validate our approach through a real case. The results and conclusions following this simulation are detailed.

Chapter 6 introduces a variant on our approach, called people-based approach. Instead of considering skills mix polls of agents, we work with each agents as an individual with his own and original skills profile.

Chapter 7, the last one, presents the conclusions that can be made following the work achieved in this project. We also discuss the future work that could be done on this project.

At the end of this thesis, there are four appendices detailing some technical points:

- In appendix A, the Scaled Conjugate Gradients Algorithm is detailed
- In appendix B, we describe the Erlang's C model
- In appendix C, we discuss in further detail the stochastic modelling of the queues
- In appendix D, the different algorithms are presented in a well commented way

Chapter 2

The feasibility study

2.1 Solution method

In this first section, we will describe the strategy used to tackle the problem we are considering.

The purpose of a scheduling method in a call centre is to predict the minimum number of agents required to achieve a certain service level and to provide a satisfactory quality of service, according to the expected activity in the call centre. In our method, the major unknowns are the connectivities between the telephone channels and the skills groups. The purpose of our method is to find the optimum values of the connectivities, i.e. the values that respond the best to the conditions described earlier.

Each of these conditions, i.e. a minimum number of agents, a certain service level, a good quality of service, ..., can be modelled as cost terms. For instance employing more agents will increase the cost term corresponding to the minimisation of the number of staff. But on the contrary, employing less agents will increase the cost term corresponding to the service level. Thus the basis of our approach is to find the minimum of an overall cost function, which is made up from different cost terms mentioned above.

Another essential part of our model is to add some constraints to the cost function. A constraint term is an additive term which prevents the connectivities from taking certain values. This additive term will have a huge value in the forbidden area of the connectivities and a small value in the authorised area. So it looks like a cost term, but is not really one as we already know how to optimise it.

In the minimisation of a cost function, we have to distinguish the local minima from the global one. The minimum for which the value of the cost function is smallest is called the global minimum while other minima are called local minima. As a consequence of the non-linearity of the cost function, it is not in general possible to find closed-form solutions for the minima. Instead, we consider algorithms which involve a search through connectivities space consisting of a succession of steps of the form:

$$C_{j+1} = C_j + \triangle C_j$$

where j labels the iteration steps and $\triangle C_j$ represents the connectivity matrix change. Different algorithms involve different choices of the connectivity matrix change.

The algorithm used to minimise our cost function involves taking a sequence of steps through connectivities space. It is convenient to consider each of these steps in two parts:

- first we must decide the direction in which to move;
- secondly we must decide how far to move in that direction.

The particular algorithm we have used is called the scaled conjugate gradients algorithm and is discussed in more detail in appendix A. For such algorithms, the cost function is guaranteed not to increase as a result of a change in the connectivities. One potential disadvantage for such algorithms is that if they reach a local minimum they will remain there forever, as there is no mechanism for them to escape (as it would require a temporary increase in the error function).

2.2 The cost function

In this section, we outline the cost function used in the study, the minimum of which corresponds to the solution of the problem. Correct definition of the cost function and constraints on acceptable solutions is the key to success in optimisation problems. For this reason, in this section, we describe our approach and assumptions in some detail. All the materials described here came from a feasibility study [1] made by Prof David Lowe and Dr Ian Nabney. However some points of this study have been modified in the current project and are consequently presented in their latest version.

2.2.1 Mathematical notation

Before we construct the various components of the cost function, we need to introduce some mathematical notation to describe the problem.

The total number of incoming queues of telephone calls is denoted by Q and at time t the call density in queue q is given by $n_q(t)$. In the feasibility study, the call density is assumed to be constant across each half hour period, which is our time basis, and we assume that the calls are uniformly distributed across the half hour period. In practice, the total number of calls offered in an half hour period, $N_q(t)$, is measured along with the mean call duration $\tau_q(t)$ in seconds in this period. Therefore to convert the measured data into the local call density (the average number of calls active at an instant) we need simply to scale the measured data as $n_q(t) = N_q(t) \times \tau_q(t)/(30 \times 60)$.

The total number of pools of agents is denoted by P. People with similar skills profiles will be grouped together into a small number of groups, where each of the skills profiles would be overlapping with other profiles. These profiles indicate the ability of members of the pools to answer any one of the Q different types of calls. Hence, when a call needs answering, the call would be routed to one of the skills mix pools for answering, even though that pool may not have a 100% competency to answer that specific call. The skills profile is measured with a Q-dimensional vector s_p, for there are Q different types of incoming calls.

We can characterise the whole workforce by forming the complete skills matrix which lists the skills profiles for every pool. An entry in this matrix, S_{pq} is the ability of a person belonging to the skills mix pool p to be able to answer a telephone call of type q, that is from queue q. For convenience (as we think of these entries as probabilities), the values are scaled to lie between zero and one. As each skills mix pool has its own skills profile, this skills matrix S is of a size $(P \times Q)$.

Finally, we consider that there is also a connectivity matrix C. The entries c_{pq} denote a link between queue q and pool p. There are various constraints that have to be imposed on this connectivity matrix to ensure that it is valid.

2.2.2 The cost function

The construction of the cost function is based on several constraints that we should impose. The variables of this cost function are the connectivities, whose final values will correspond to the optimum value of the cost function.

One of these constraints is that, at an instant, there are only $n_q(t)$ calls active in queue q. Hence the sum of all connectivities across all pools must be less than or equal to this, i.e. $\sum_{p=1}^{P} c_{pq} \leq n_q(t)$. As we wish to solve the problem (i.e. answer the telephone calls) whilst using as few people as possible, then another constraint on the connectivity matrix is that the sum of all connectivities should be reduced as far as possible, i.e. $\sum_{p=1}^{P}\sum_{q=1}^{Q} c_{pq}$ should be minimised. Finally we should ensure that the connectivities are set so that the most qualified agent available will answer the call. For this purpose, we need to introduce a measure of the quality of service provided to the customer and consequently a new constraint which is to maximise this measure.

The measure of the quality of service tells us how well the solution matches the problem, i.e. how many calls are answered by people of high competencies. In the traditional statistical case this takes the form of what fraction of calls are answered on average within a set time period, say 30 seconds. In the traditional approach, this statistical reference is sufficient because all the calls are assumed to be answered by completely qualified people. In our approach we still want to answer the highest possible fraction of incoming calls, but we also have to take into account how well they are answered, because they can be answered by people without all the required skills. A straight forward way of measuring the quality of service provided to calls from queue q is to use the following formula: $\sum_{p=1}^{P} c_{pq}S_{pq}$. If all the calls are answered by people with a skill of 100%, then this measure is maximised to $n_q(t)$. Hence, a measure of total quality of service across all queues, suitably normalised would be

$$\frac{1}{Q}\sum_{q=1}^{Q}\frac{1}{n_q}\sum_{p=1}^{P} c_{pq}S_{pq}$$

As we are seeking to minimise a cost function, we are interesting in quantifying a loss in quality which we wish to reduce. So we may use the above definition of quality to define a measure of loss in quality as:

$$E_0 = \frac{1}{Q}\sum_{q=1}^{Q}\left(\sum_{p=1}^{P} c_{pq}S_{pq} - n_q(t)\right)^2$$

This term would be equal to zero if all the calls are answered by people with 100% competencies.

Another constraint is that there are only $n_q(t)$ calls in queue q at time t. The penalty term which represents this requirement is:

$$E_1 = \frac{1}{Q}\sum_{q=1}^{Q}\left(\sum_{p=1}^{P} c_{pq} - n_q(t)\right)^2$$

The penalty term which tries to minimise the total number of staff required is:

$$E_2 = \frac{1}{Q}\sum_{q=1}^{Q}\sum_{p=1}^{P}(c_{pq})^2$$

Finally one obvious constraint not yet considered is that negative connections are not allowed. So we need to introduce an explicit penalty term which forces the connectivities to only take on positive values. There are several ways we could choose to encode this constraint, e.g. by employing a 'potential barrier' term which 'repels' the parameters value away if it tries to go negative. More detail on this issue can be found in the section 3.3.1. The type of penalty term chosen could be:

$$E_3 = \frac{1}{Q}\sum_{q=1}^{Q}\sum_{p=1}^{P}\frac{1}{1+\exp(-\eta c_{pq})}$$

where η is a large value which induces a strong gradient barrier around the origin. However this is not a sufficiently 'strong' way to implement this constraint. A more efficient way to do it is to use the penalty function method, whose use is explained and justified in the next chapter, section 3.2.1. The term c_{pq}^2 will be added to the cost function only if the constraint is violated, i.e. the connectivity c_{pq} is negative. This term gets bigger as the connectivity c_{pq} becomes 'more negative' and is null, and so as no effect, if the connectivity is positive. As we are minimising the cost function, this last term will prevent connectivities from being negative.

The last constraint is that no call should be allocated to agents belonging to a pool with 0% competency to answer it. To prevent any call from being answered by an agent completely unqualified, we must implement the following constraint: if S_{pq} is equal to zero the corresponding connectivity c_{pq} should be null as well. An easy and efficient way to implement this constraint is to use the penalty function method, whose use is explained and justified in the next chapter, section 3.2.1. We then only have to add a penalty term c_{pq}^2 to the cost function if the constraint is violated, i.e. for the same queue q and the same pool p, S_{pq} is equal to zero and c_{pq} isn't.

Minimising a function subject to constraints is much harder than unconstrained minimisation. We want to minimise E_2 subject to constraints. By rewriting these constraints as additional costs, we can use the method of Lagrange multipliers to recast the problem as one of unconstrained minimisation. The final cost function we are interested in is obtained by combining all the above terms as follows:

$$E = E_0 + \alpha E_1 + \beta E_2$$

where α and β are Lagrange multipliers which may be set to specific values which reflect our relative importance of satisfying each of the penalty terms.

This function is the one to minimise, and to which we apply the penalty function method in order to satisfy the last two constraints described above.

2.2.3 Observations

Although a number of assumptions was made, the proposed approach is quite general and could be applied to any call centre. For the end user, the only requirement is that they should be able to specify the skills of different groups of their staff members. Once this is done, the optimisation algorithm will determine how many staff members from each profile are required.

The method operates on an instantaneous basis, i.e. the process produces the expected number of agents required in each skills mix which should be capable of answering the average number of calls in each queue in each half hour period. Each half hour period requires an separate optimisation. Because this is generally a nonlinear optimisation process, the algorithm requires an initialisation of the parameters which are then adjusted until a minimum is found. It might be the case that a minimum is found which is unsuitable. Hence one generally needs to run the optimisation algorithm several times with different random start values for the connectivities. We have found that in this problem a suitable solution is found without requiring many random starts. The results presented next were obtained using a single run in each half hour period, and using the scaled conjugate gradients optimisation method, which is detailed in appendix A.

The key assumptions made during the feasibility study are:

1. The quality of service is measured using the skill score. The time taken to answer the call is assumed to be constant. This is appropriate for circumstances where the skill corresponds to the probability of converting a call into a sale, but not for technical calls. For example, when answering technical queries, a less skilled operator may come up with the same answer in a longer period of time.

2. The staff all cost the same, which is not likely to happen for high skilled staff will earn higher wages than the low skilled operator. However it is relatively straight forward to associate different costs with different skills profiles and minimise cost, rather than staff number.

3. In each skills mixes the pool of people is assumed to be unlimited. This may not be appropriate, as some highly qualified profiles may only have a small number of operators available.

4. The calls are uniformly distributed across the half hour period. The traditional approach models the calls with an Erlang process, which results in assigning more than the expected number of operators to each queue in order to allow for fluctuations in the call density during the measured period.

As some of this assumptions need to be investigated, some tests to be make and some simulations to be run, the feasibility study will be held further in this current project. Thus the results obtained and the conclusions drawn will be closer to the actual behaviour of a call centre.

2.3 Results

The results presented in this section are derived from actual data provided by CallScan. This data described the activity of a call centre for one complete week. Data was provided on call load and duration from six different queues throughout the whole week, that is six days (Monday to Saturday) with calls logged every half an hour from 08:30 to 18:00 each day. The call density is displayed on figure 2.1, for each queue and for each half hour period of the week.

On the basis of this information, the statistical model produces an estimate of the number of agents needed to be available in each half hour period throughout the week in order to answer these calls. Figure 2.2 shows the estimated staff numbers required for the same period, based on the assumption of a 95% service level objective within 30 seconds. This figure is a benchmark for comparing the results of the optimisation approach against.

To proceed with the minimisation of the cost function, we need to determine specific skills profiles. Table 2.1 described the 4 broad skills mixes chosen from the provided data.

Using significantly many more than just 4 skills profiles, an almost perfect solution to the problem can be found, in the sense that it is always possible to answer all calls with people with a skill of 100. Hence we deliberately chose a small number of profiles in order to make the problem more realistic and to demonstrate the effectiveness of the approach. It is worth noting that the third skills mix area is less qualified than the other ones and therefore one can expect the agents from this skills mix area not to be

Skills Mix Matrix (unscaled)

Queues	1	2	3	4	5	6
Skill 1	35	100	100	74	0	0
Skill 2	36	0	0	0	100	44
Skill 3	60	0	60	0	0	0
Skill 4	80	0	51	30	100	0

Table 2.1: The four skills mix factor used in the experiments reported. Note the obvious inferiority of skills area number 3.

employed in providing the service. This is not the case, as we will show.

Figure 2.3 displays the aggregated final results of the proposed optimisation approach, compared to the total number of agents required according to the traditional approach. The obvious point about this figure is that the neurally-inspired approach produces estimates of the required number of agents which are significantly less than estimates produced by the standard model. As mentioned previously, use was made of staff from the obviously less qualified skills profile in order to achieve this staffing level. This can be seen in Figure 2.4 which shows the breakdown of the data in Figure 6 into the predicted number of agents required in each of the four skills mixes.

The two other points that need now to be examined alongside the number of staff are what proportion of all calls were answered and how well those calls were answered according to our quality index. Figure 2.5 depicts the percentage of calls answered throughout the period by the approach used in this report. As can be seen, over 99% of all calls have been answered, despite using fewer staff than the traditional approach. Figure 2.6 shows the quality of service provided by the new approach. If all calls were answered by people with 100% competencies then the service quality index would be equal to unity. However if fewer than 100% calls were answered, or if calls were answered by people with inferior competencies, then the quality index should be reduced. This figure demonstrates that a high level of service quality is provided despite using

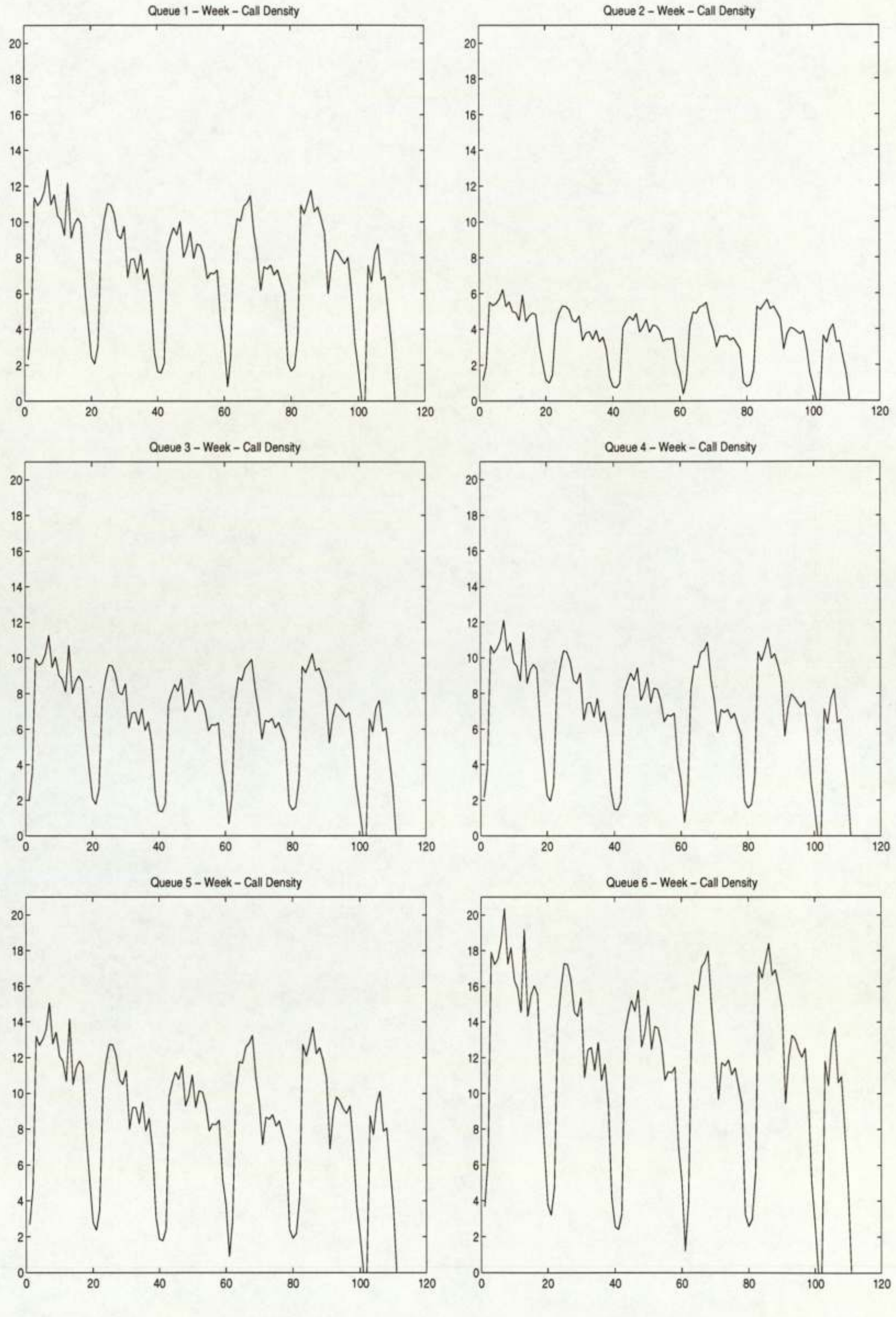


Figure 2.1: The call density in each of the six queues. The call density is the number of calls received in each half hour period, multiplied by the average call duration and then divided by the number of seconds in the half hour period.

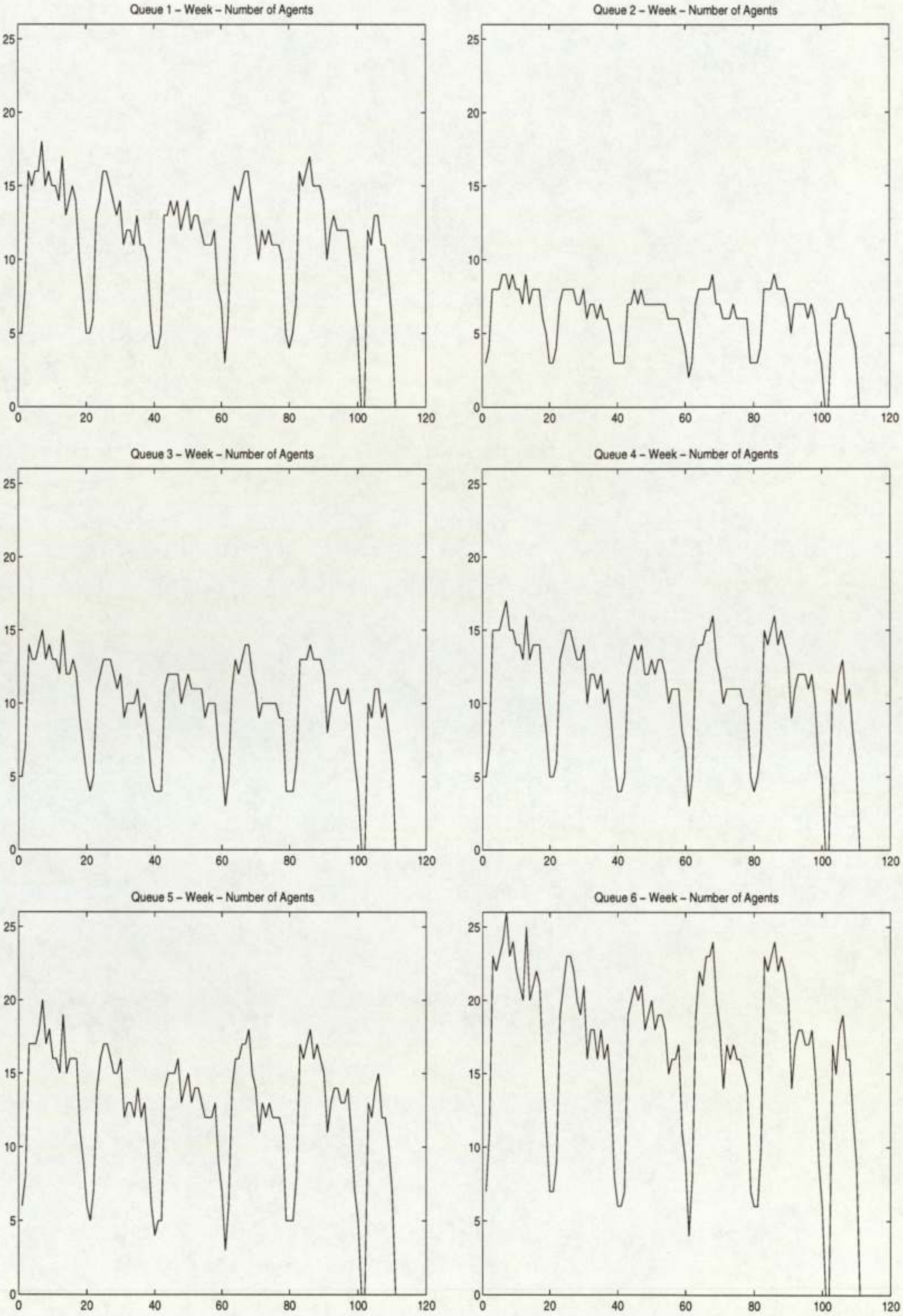


Figure 2.2: Number of agents used in each of the six queues according to the standard model.

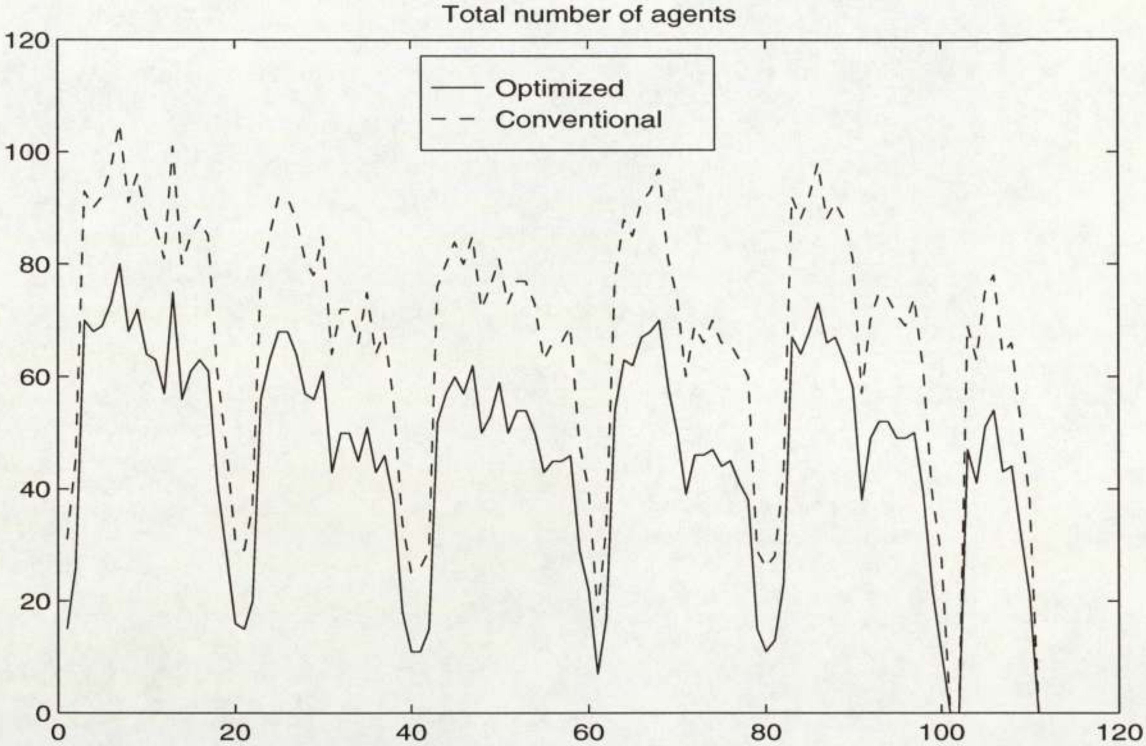


Figure 2.3: Comparison between the number of agents used in the traditional model (the dotted line) and the number of agents used in our approach (solid line).

fewer agents and also using agents with restricted competencies.

A last point needs to be evocate: it is the time taken by this optimisation algorithm. All optimisation results using the skills mix approach were obtained in a matter of a couple of minutes.

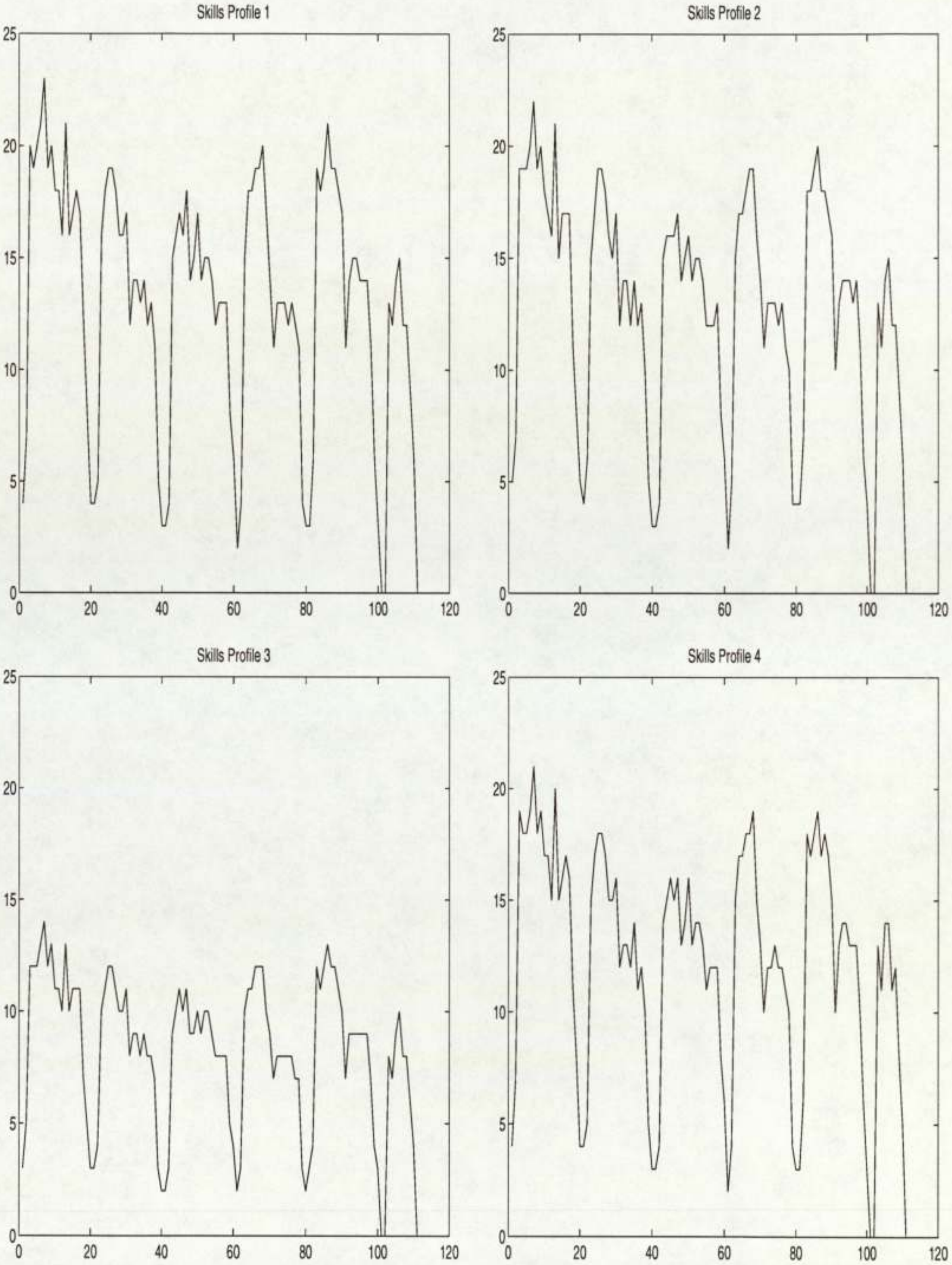


Figure 2.4: Predicted number of agents in each skills profile.

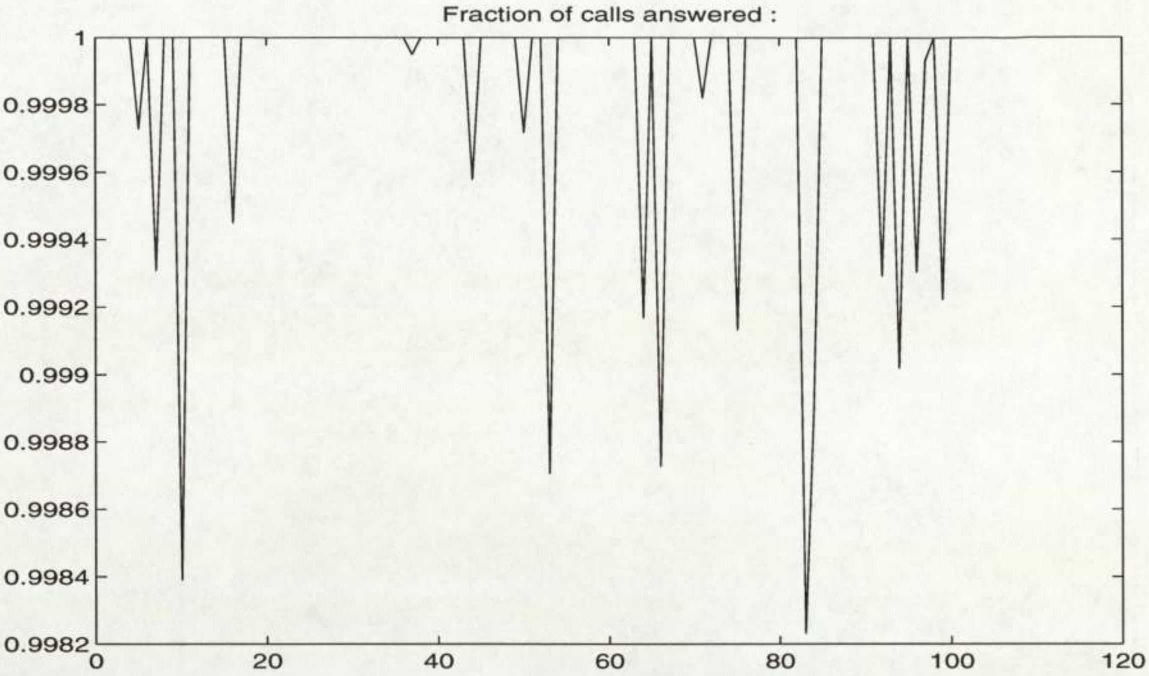


Figure 2.5: The percentage of all calls answered by the optimisation method.

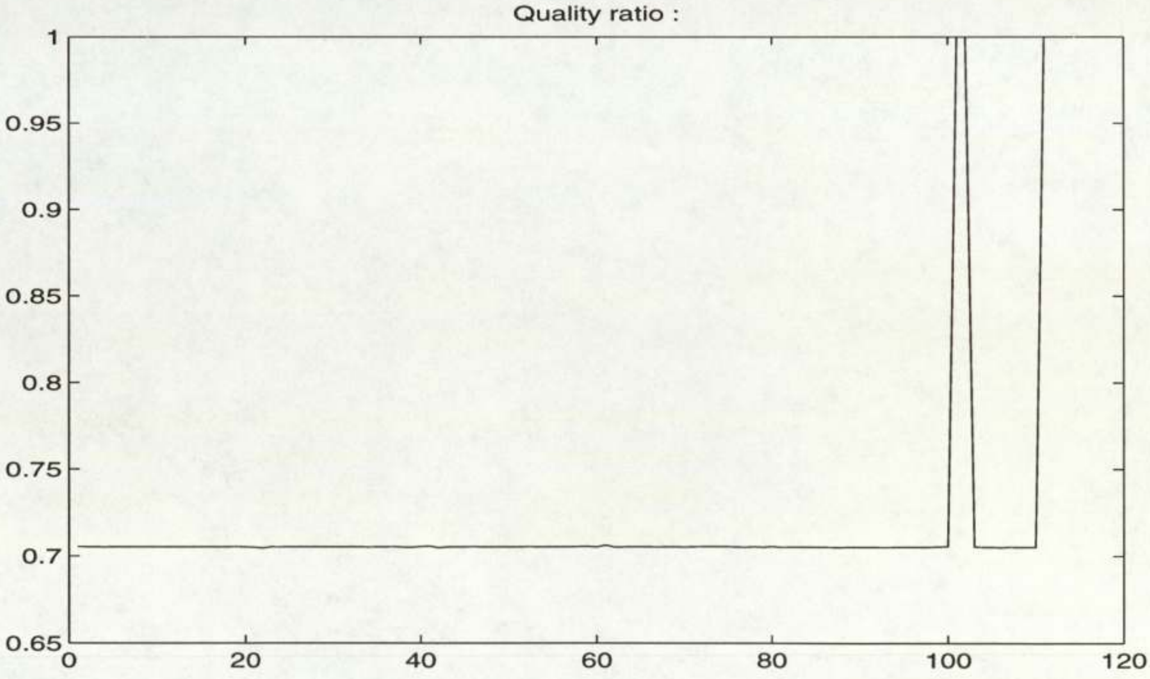


Figure 2.6: The quantification of the quality of service provided by the skills mix approach method. On this scale a value of 1 indicates perfect quality.

2.4 Conclusions

In this chapter, we have described a method for obtaining a solution to the resource allocation problem of matching agents to telephone call demand services. The main advantage of this new approach is that the potential agent pool is considered as heterogeneous, differentiated by a coarse set of overlapping skills mix profiles. In theory any call could be answered by any person in any one of the skills mix areas. Our method returns the predicted number of agents required in each pool. The proposed approach appears to predict fewer staff required, compared to the traditional method. The overall performance in terms of percentage of calls answered reaches a good level. The method also returns a quantitative index of the quality of service provided, which may also be monitored.

The following points weren't considered in this feasibility study, some of them will investigated further in this report.

1. No algorithm was defined for allocating agents to calls. A simple method, such as assigning the best qualified agent available to the call, would probably be adequate.

2. No attempt was made at temporal smoothing. The method works on instantaneous half hour values and does not account for shift patterns - it is only driven by the demands of the call density in each queue.

3. Only one type of quality of service was considered. There may be a more appropriate definition which embodies more information specific to the problem domain.

4. Alternatives exist to encode some of the constraints in this report which may lead to more robust cost functions.

5. We have assumed that all staff cost the same. It would be relatively straight forward to associate different costs with different skills profiles and minimise costs, rather than staff numbers.

6. We have assumed that there is a unlimited pool of people in each skills mix. This will not be appropriate for all the more highly qualified profiles.

7. We have assumed that the calls are uniformly distributed across the half hour period. With more information on the actual time of arrival of each call, it would be possible to take into account fluctuations in the call density within each period.

8. We have assumed that each call in a given queue takes the same time to be answered, that is the average service time for all the calls from that queue. That is quite unrealistic, as the real times to answer calls from a given queue vary around this average value. Therefore it would be interesting to model the distribution of the incoming calls.

Chapter 3

Capping the number of agents

3.1 Introduction

So far we have assumed that there is an unlimited pool of people in each skills mix. This may not be appropriate, as some highly multi-skilled profiles may only have a small number of operators available. This leads us to take into account a new constraint: in highly qualified profiles, the number of agents available is limited by a capping number. This constraint could be characterised as a hard constraint, for it should not be broken. Typically, if a call centre could not allocate more than a certain number of agents to a pool, the result given by our algorithm should not exceed this and only approach the capping number, but should respect absolutely the constraint. A soft constraint is, for instance, to minimise the number of staff, and then we don't have to respect some tough criterion. Therefore we had to use some strong method which gives us results that respect the constraint. A good method to use here is the penalty function method.

3.2 Use of the penalty function method

The purpose of this section is to describe the penalty function method, to explain and justify its use in the current problem.

3.2.1 The penalty function method

Suppose we are minimising a function $f(x)$ subject to the vector c of constraints:

$$c_i(x) = 0 \quad \text{or} \quad c_i(x) \geq 0 \qquad i = 1, ..., n$$

We can construct a quadratic penalty term $\frac{\delta}{2}\widehat{c}\ \widehat{c}^T$, where $\widehat{c}$ contains the constraints violated at x and δ is called the penalty parameter. We add this penalty term to the function we are minimising to obtain a new function:

$$P_Q(x, \delta) = f(x) + \frac{\delta}{2}\widehat{c}\,\widehat{c}^T$$

Now the problem is reduced to an unconstrained minimisation. The penalty term is continuously differentiable, but has discontinuous second derivatives where inequality constraints are exactly satisfied.

An important point in this method is the choice of δ. We denote $x^*(\delta)$ the unconstrained minimum of $P_Q(x, \delta)$. Under mild conditions $\lim_{\delta \to \infty} x^*(\delta) = x^*$, the true minimum with constraints. So the way to proceed can be described as following:

- minimise $P_Q(x, \delta)$
- increase δ
- iterate until convergence

The penalty parameter δ can't be chosen too large initially.

The effect of adding the penalty term is to create a local minimum near x^* for sufficiently large values of δ. The example described in the next subsection makes it clearer.

3.2.2 Example

The following example illustrates the use of the penalty function method.

We want to minimise $f(x) = x^3$, subject to the constraint $x \geq -1$. The quadratic penalty term is $\frac{\delta}{2}(x+1)^2$ and the new function to minimise is:

$$P_Q(x, \delta) = x^3 + \frac{\delta}{2}(x+1)^2$$

The different functions are drawn on Figure 3.1. The solid line represents the new function to minimise, $P_Q(x, 12)$, we have chosen $\delta = 12$ for this example. The dashed line represents the function to minimise under constraint, $f(x) = x^3$. The dash-dot line represents the quadratic penalty term, $(x+1)^2$.

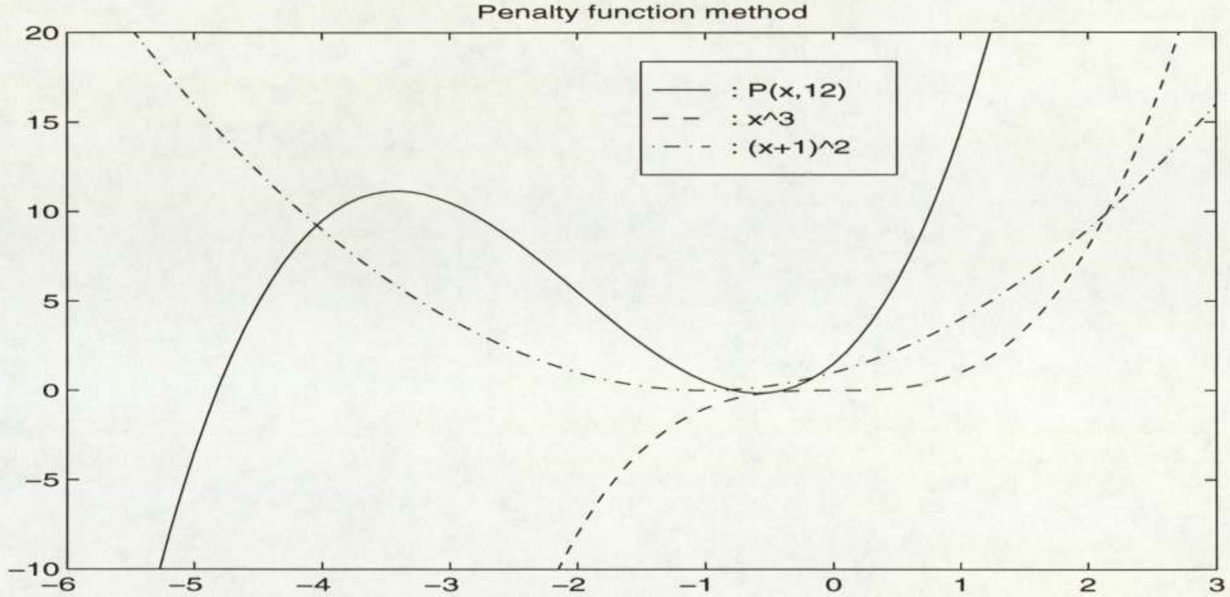


Figure 3.1: The penalty function method.

3.2.3 Application

In our current problem, we have to minimise the cost function derived in the previous chapter subject to the new constraint we have introduced just at the beginning of this section. This constraint is that in given pools of agents the number of people can't be greater than a fixed number, the capping number. This constraint can be written as following:

$$\sum_{q=1}^{Q} c_{pq} \leq A_p \tag{3.1}$$

where A_p is the maximum number of agents available for the pool p. The total number of agents in the pool p is equal to the sum of all the connectivities linking the Q queues to the pool p, that is $\sum_{q=1}^{Q} c_{pq}$.

We can rewrite the inequality 3.1 in order to use it in the penalty function method:

$$A_p - \sum_{q=1}^{Q} c_{pq} \geq 0$$

Thus, if the constraint is violated, i.e. $\sum_{q=1}^{Q} c_{pq} > A_p$, we add to our cost function the following penalty term:

$$\frac{\delta}{2}\left(A_p - \sum_{q=1}^{Q} c_{pq}\right)^2$$

By adding this constraint, we slightly slow the algorithm, but the running time of this algorithm remains quite reasonable. All optimisation results are obtained in a matter of several minutes.

3.3 Other approaches

In this section we will describe the other approaches to capping the number of agents in certain skills mixes that we have tried and why they were abandoned.

3.3.1 Use of a step function

The idea behind this approach is to add a term which will take on big values if the number of agents in the considered groups is bigger than the number authorised and small values else. As we are minimising an overall cost function, this additive term will keep the connectivities in the right area. i.e. where they are such that the numbers of agents are capped. A function whose value increases, or decreases, suddenly when its variables exceed a certain value, is called step function. We will describe how such a function has been tried in our approach.

The sigmoid function

The sigmoid function is a widely used step function, that can be defined as follows:

$$f(x) = \frac{1}{1 + \exp(-\eta(x - \alpha))}$$

If x is greater than the step value α, then $f(x)$ goes towards one. If x is less than the step value α, then $f(x)$ goes towards zero. Of course we can add a multiplicative coefficient in front of the function f so that the function takes on big values. The role of the coefficient η is to increase the steepness of the function around the step value α. The greater η is, the steeper the curve of the function is, when it goes from zero to one, that is around α. Figure 3.2 illustrates more clearly the properties of this function, with the particular function:

$$f(x) = \frac{1}{1 + \exp(-6 \times (x - 5))}$$

Results and conclusions

The constraint that we are considering is the following one:

$$\sum_{q=1}^{Q} c_{pq} \leq A_p$$

So the sigmoid function should take as a variable $\sum_{q=1}^{Q} c_{pq}$ and as a step value A_p. The coefficient η is chosen sufficiently large so that the step of the sigmoid function is steep enough. We add to the current cost function the following term:

$$\frac{1}{1 + \exp(-20 \times (\sum_{q=1}^{Q} c_{pq} - A_p))}$$

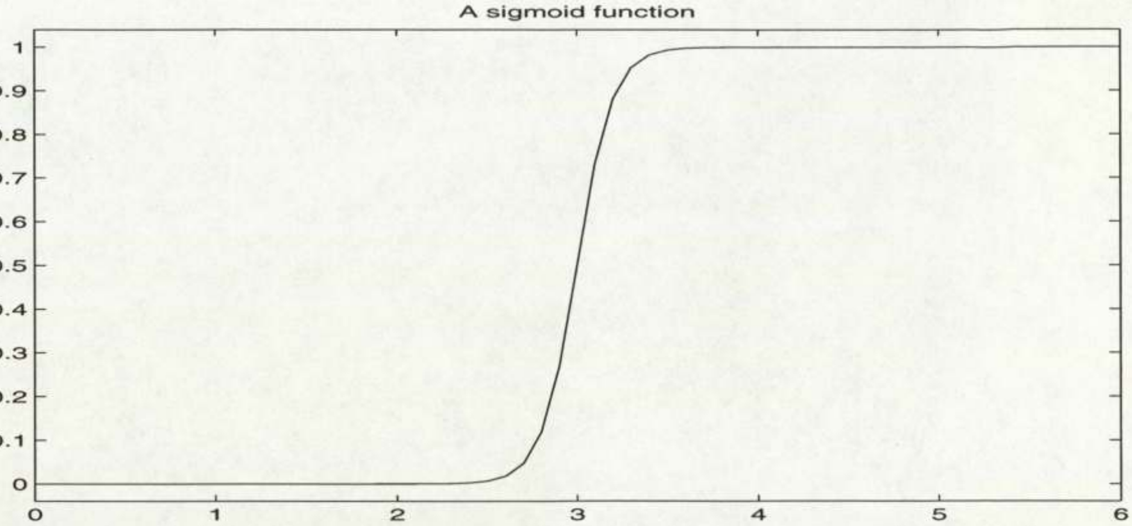


Figure 3.2: This figure illustrates the properties of the sigmoid function. The step value α is equal to 3 and the coefficient η is equal to 10.

The effect of adding this term is to increase the final value of the cost function, if the constraint is violated. As we are trying to minimise the cost function, the values of the connectivities obtained should satisfy this constraint.

The results given by this method are not completely satisfactory. If the constraint is to harsh, i.e. the capping number A_p is to small, the connectivities obtained don't satisfy strictly the constraint. Their sum on each queue, i.e. $\sum_{q=1}^{Q} c_{pq}$, is close to, but not less than A_p. This can be explained by the fact this constraint is not strong enough compared to the two other ones tried, the penalty function method and the softmax method. However, with this method, the algorithm runs quite fast and is barely slowed by the added constraint.

3.3.2 Use of the softmax method

With the softmax method [8], the basic idea is to make a change of variables, so that the new ones can't break the constraint. So we keep the same cost function, but we rewrite the variables of this function, so that they always satisfy the constraint.

The softmax method

Let's explain the softmax method with a simple example. We consider a function E, whose variables are denoted x_i, for $i = 1, ..., n$. We want to minimise E, subject to the constraint:

$$\sum_{i=1}^{n} x_i \leq A$$

Now, instead of working with the variables x_i, we will consider the variables y_i, which are defined so that they always satisfy the constraint. Thus $\sum_{i=1}^{n} y_i \leq A$ is always satisfied. The new variables y_i are defined by $y_i = f(x_i)$. We now have to find the function f.

We want that $\sum_{i=1}^{n} f(x_i) \leq A$, which is equivalent to $(A - \sum_{i=1}^{n} f(x_i)) \geq 0$. Then we introduce a new variable, a slack variable, $\bar{A} = A - \sum_{i=1}^{n} f(x_i)$. This new variable is subject to the constraint: $\bar{A} \geq 0$. We can now define f, so that $\sum_{i=1}^{n} f(x_i) = A - \bar{A}$. This can be done by defining f in the following way:

$$\forall \quad 1 \leq i \leq n \quad f(x_i) = \frac{(A - \bar{A}) \exp(x_i)}{\sum_{i=1}^{n} \exp(x_i)}$$

So the new problem can be summarised:

- the new function to optimise is $E(f)$, i.e. the function E with the new variables $y_i = f(x_i)$
- the function f is defined as described in the previous paragraph
- there is a new variable and a new constraint: $\bar{A} \geq 0$

Results and conclusions

We can easily apply the softmax method to our approach. The constraint that we are considering is:

$$\sum_{q=1}^{Q} c_{pq} \leq A_p$$

For each pool p, in which we are trying to minimise the number of agents, we will have to introduce a softmax function f_p and a new variable $\bar{A}_p$. The softmax function is defined as follow:

$$\text{For} \quad p = 1, ..., P \quad f(c_{pq}) = \frac{(A_p - \bar{A_p}) \exp(c_{pq})}{\sum_{q=1}^{Q} \exp(c_{pq})}$$

And the new variable $\bar{A}_p$ is equal to $\bar{A}_p = A_p - \sum_{q=1}^{Q} f(c_{pq})$. This variable should be positive, so it is subject to the constraint $\bar{A}_p \geq 0$. A straight forward way to implement this last constraint is to use a sigmoid function as described before.

This method has a huge drawback, that it is very computationally expensive. To optimise a week of incoming calls, it takes one hour and a half to obtain the required number of agents in each pool, with only one constraint on the number of staff. It is 30 times longer than with the sigmoid function. Furthermore, if we want to implement constraints on the number of people in more than one pool, then it will be hugely long. This situation is due to the calculations of the derivatives of the softmax function, which are very complicated.

With this method the constraints are satisfied, as with the penalty function method. But the results obtained are not better than the ones given by the penalty function in terms of quality of service provided to the customers and in terms of service level. So it is not worth keeping this method.

3.4 Results and Conclusions

Our algorithm now needs new parameters: the capping numbers for each pool of agents. As there are four skills mixes in our test data, we need four capping numbers, one for each half hour period. Of course, as there may be no need to cap one or several pools of agents, the capping numbers are not compulsory. So the capping numbers can be presented to the algorithm as a (120×4) matrix: four capping numbers for each one of the 120 half hour periods of the week. It may also be presented as a vector with 4 elements, in case there is a single overall capping number for all the half hour periods.

If we assume that there is an unlimited pool of people in each skills mix, the predicted numbers of agents required for each half hour period are:

- A_t in skills mix 1
- B_t in skills mix 2
- C_t in skills mix 3
- D_t in skills mix 4

These numbers are obtained for each half hour period during a whole week. So there are stored in a (120×4) matrix.

We are now going to cap the numbers of agents in the skills mixes. These are the assumptions and notations useful to read the tables detailing the results:

- We have assumed that the calls are uniformly distributed across the half hour period.
- The agents required in skills mix 3 are less qualified than the ones required in the other skills mixes, so they are assumed to be less expensive. Therefore the capping in this skills mix could be weaker than in the other ones.
- In all the tables, the first line contains the uncapped results.
- Staff denotes the sum of all the agents required in all the half hour periods for the whole week.
- S.L. denotes the service level.
- <0.95% denotes the number of half hour periods for which the service level is less than 95% of calls answered within 30 seconds.
- Q.R. denotes the quality ratio.

3.4.1 First example

The capping numbers for each half hour period are obtained by decreasing the uncapped results for each half hour period in each skills mix.

Skills				Staff	S.L.	<95%	Q.R.
A_t	B_t	C_t	D_t	5085	0.9999	0	0.7347
$A_t - 1$	$B_t - 1$	$C_t - 1$	$D_t - 1$	4684	0.9895	2	0.7045
$A_t - 2$	$B_t - 2$	$C_t - 2$	$D_t - 2$	4380	0.9297	87	0.6545
$A_t - 3$	$B_t - 3$	$C_t - 3$	$D_t - 3$	4164	0.8752	95	0.6424

Table 3.1: Results of the first example.

As a consequence of the decrease in the number of staff in all the pools, the service level and the quality ratio decrease.

3.4.2 Second example

The capping numbers for each half hour period are obtained by decreasing the uncapped results for each half hour period in skills mixes 1, 2 and 4. In skills mix 3, the capping number for each half hour period is equal to the uncapped result.

Skills				Staff	S.L.	<95%	Q.R.
A_t	B_t	C_t	D_t	5085	0.9999	0	0.7347
$A_t - 1$	$B_t - 1$	C_t	$D_t - 1$	4792	0.9964	0	0.7055
$A_t - 2$	$B_t - 2$	C_t	$D_t - 2$	4474	0.9617	23	0.6765
$A_t - 3$	$B_t - 3$	C_t	$D_t - 3$	4245	0.9098	90	0.6584

Table 3.2: Results of the second example.

By not capping the number of agents in skills mix 3, we obtain slightly better results for the service level and the quality ratio.

3.4.3 Third example

The capping numbers for each half hour period are obtained by decreasing the uncapped results for each half hour period in skills mixes 1, 2 and 4. Whereas in skills mix 3, the capping number for each half hour period is obtained by increasing the uncapped result.

Skills				Staff	S.L.	<95%	Q.R.
A_t	B_t	C_t	D_t	5085	0.9999	0	0.7347
$A_t - 1$	$B_t - 1$	$C_t + 1$	$D_t - 1$	4826	0.9995	0	0.7037
$A_t - 2$	$B_t - 2$	$C_t + 2$	$D_t - 2$	4688	0.9898	2	0.6831
$A_t - 3$	$B_t - 3$	$C_t + 3$	$D_t - 3$	4502	0.9684	20	0.6607
$A_t - 4$	$B_t - 4$	$C_t + 4$	$D_t - 4$	4376	0.9418	77	0.6314

Table 3.3: Results of the third example.

We compensate for the loss in staff in skills mixes 1,2 and 4 by adding agents in skills mix 3. Therefore the service level and the quality ratio don't decrease as much as previously.

3.4.4 Fourth example

The capping numbers for each half hour period are obtained by decreasing the maximum of the uncapped results for all the half hour periods of the whole week.

The maximum of the uncapped results in each pool are denoted by A, B, C and D:

$$A = max(A_t) \qquad B = max(B_t)$$
$$C = max(C_t) \qquad D = max(D_t)$$

Skills				Staff	S.L.	<95%	Q.R.
A_t	B_t	C_t	D_t	5085	0.9999	0	0.7347
$A-1$	$B-1$	$C-1$	$D-1$	4999	0.9998	0	0.7344
$A-2$	$B-2$	$C-2$	$D-2$	4976	0.9996	0	0.7341
$A-3$	$B-3$	$C-3$	$D-3$	4926	0.9988	1	0.7329
$A-4$	$B-4$	$C-4$	$D-4$	4896	0.9953	5	0.7308
$A-5$	$B-5$	$C-5$	$D-5$	4827	0.9886	8	0.7271
$A-6$	$B-6$	$C-6$	$D-6$	4785	0.9754	20	0.7221
$A-7$	$B-7$	$C-7$	$D-7$	4727	0.9591	26	0.7141

Table 3.4: Results of the fourth example.

In this example, the capping works only for the half hour periods where the number of staff required is high. So it doesn't affect too much the service level and the quality ratio.

3.4.5 Fifth example

The number of agents in skills mix 3 is uncapped, whereas in skills mixes 1, 2 and 4 the capping numbers for each half hour period are obtained by decreasing the uncapped results.

Skills				Staff	S.L.	<95%	Q.R.
A_t	B_t	C_t	D_t	5085	0.9999	0	0.7347
$A_t - 1$	$B_t - 1$	/	$D_t - 1$	4886	0.9999	0	0.7169
$A_t - 2$	$B_t - 2$	/	$D_t - 2$	4875	0.9996	0	0.7038
$A_t - 3$	$B_t - 3$	/	$D_t - 3$	4870	0.9994	0	0.6965
$A_t - 4$	$B_t - 4$	/	$D_t - 4$	4866	0.9990	0	0.6931
$A_t - 5$	$B_t - 5$	/	$D_t - 5$	4861	0.9989	0	0.6834
$A_t - 6$	$B_t - 6$	/	$D_t - 6$	4852	0.9986	0	0.6754
$A_t - 7$	$B_t - 7$	/	$D_t - 7$	4849	0.9983	0	0.6714
$A_t - 8$	$B_t - 8$	/	$D_t - 8$	4843	0.9980	0	0.6692

Table 3.5: Results of the fifth example.

As the number of agents in skills mix 3 is left uncapped, the loss in staff in skills mixes 1, 2 and 3 corresponds to an increase in skills mix 3. Thus the service level remains high, whereas the quality ratio decreases because of the poor competencies of the agents in skills mix 3.

3.4.6 Conclusions

First some remarks need to be made. The time taken by the optimisation algorithm becomes longer because of the added constraint, but remains in a matter of few minutes. The second point is the starting values of the algorithm. As this optimisation algorithm is a nonlinear process, it might require several runs with different random start values for the connectivities. However we have found that, with the capping constraint as well as without it, a suitable solution is found without requiring many random starts.

We have tried three different methods of capping the number of people in certain pools:

- a sigmoid function, whose results are not quite satisfactory although it is quite fast
- the softmax method, which is efficient but really too long to run
- the penalty function method, which is as efficient as the softmax method and much faster

A lot of experiments can be done with this method. But the aim of this method is to decrease the number of people in certain skills mixes, which are the most qualified and therefore the most expensive. We have found that it is possible to decrease the number of agents in the most qualified pools and still maintain a good service level. This can be done by increasing in the same time the number of agents in less qualified pools, which also leads to a decrease in the quality of service provided.

According to the five examples described earlier in this chapter, the best way to apply this is to cap the number of people in highly qualified pools and to let uncapped some less qualified pools. Thus the total number of staff won't decrease too much, which enables us to keep a good service level, whereas the quality of answer might decrease. Some economy can then be done as the more highly qualified staff are assumed to be more expensive, but the measure of the quality of service must be carefully observed in order to maintain a satisfactory quality of service.

Chapter 4

Stochastic modelling of the queues

4.1 Describing a queueing system

In this section we will justify and describe the use of a stochastic model of the queues. We shall also describe the most general way to model a queueing system in order to present the points that have guided our research. In fact, the researches made to design a stochastic model of the queues of our system are based upon the queueing theory. That's why it is first essential to introduce some general notions about queueing theory, [2] [3] [4] [5] [6] [7], before developing our real works. For further details about the traditional way of modelling the queues in a call centre, i.e. the Erlang's C model, see appendix B.

4.1.1 Introduction

So far we have assumed that the calls are uniformly distributed across the half hour period, which is equivalent to consider the call density constant within each period. This assumption is quite inaccurate to describe the actual behaviour of the queues, because we don't account for any fluctuations in the call density. With the use of a stochastic model of the queues, it would be possible to take into account fluctuations in the call density which results in assigning more than previously predicted number of agents to each pool.

In this new approach, we still use the call volume, i.e. the number of incoming calls, and the mean call duration in an half hour period as the only two parameters. We now want to build a model that accounts, first, for fluctuations in the time between call arrivals and, secondly, in the service time. Non-uniform call arrivals cause clustering in the arrivals, as there may be busy periods inside an half hour as well as quiet ones. It is much more realistic than considering each queue as a steady stream of calls to the call centre. The obvious consequence of this new assumption is that our model will require more staff in order to cope with the clusters of calls that may arrive. The second aspect in this new approach is to consider the service time as a random variable, and no more as a constant number. This means that the service time will take on random values from a given distribution. Consequently, some calls will take more time than others to answer, and especially more time than the mean value of the call duration. This also leads to increase the number of agents, in order to compensate the unavailability of some agents who are answering particularly long calls.

First we will present the elements of a queueing system. Customers from a population or source enter a queueing system to receive some type of service. In our case a customer will be an incoming phone call. The service facility of the queueing system has one or more servers. A server is an entity capable of performing the required service for a customer, so we will consider each agent as a server. If all servers are busy when a customer enters the queueing system, the customer must join the queue until a server is free. So in a queueing system, it is usual to consider one queue of incoming calls.

4.1.2 Queueing theory notations and definitions

First it is necessary to define and precise the notations, which we will use in this chapter.

The variable t is considered as a continuous variable, i.e. a genuine measure of the time, and not as an index for the half hour periods as in earlier chapters.

In the usual treatment of queueing theory, only one queue is considered and so it is in this section. Up to now, q has been used to index the multiple incoming queues. In order to avoid the confusions, all the variables related to the single queue considered in the queueing model are denoted with d. The basic queueing theory notations and definitions are listed below:

- c: number of identical servers
- d: random variable describing the time spent in queue
- L: expected steady state number of calls in the system, $E[N]$
- L_d: expected steady state number of calls in the queue, $E[N_d]$
- L_s: expected steady state number of calls receiving service, $E[N_s]$
- λ: average arrival rate of calls to the system
- μ: average service rate per server, that is the average rate of service completions while the server is busy
- $N(t)$: random variable describing the number of calls in the system at time t
- N: random variable describing the steady state number of calls in the system
- $N_d(t)$: random variable describing the number of calls in queue at time t
- N_d: random variable describing the steady state number of calls in queue
- $N_s(t)$: random variable describing the number of calls receiving service at time t
- N_s: random variable describing the steady state number of calls receiving service
- $P_n(t)$: probability that there are n calls in the system at time t
- p_n: steady state probability that there are n calls in the system

- ρ: server utilisation = $\lambda/(c \times \mu)$
- s: random variable describing the service time
- τ: random variable describing inter-arrival time
- w: random variable describing the total time spent in the system
- W: expected steady state time in the system, $E[w]$
- W_s: expected steady state service time, $E[s]$
- W_d: expected steady state time in queue, $E[d]$

There are some obvious relationships between some of the random variables listed above. They depend upon the two states in which the system might be.

The system is said to be in the transient state if the number of calls in the queue and in service depends strongly upon both the initial conditions and upon how long the system has been in operation:

$$N(t) = N_q(t) + N_s(t)$$
$$w = q + s$$

After the system has been in operation for a long time, the influences of the initial conditions and of the time since start-up have 'damped-out' and the number of calls in the queue and in service is independent of time. The system is then said to be in the steady state:

$$N = N_q + N_s$$
$$L = L_q + L_s$$
$$W = W_q + W_s$$

The population of customers is assumed to be infinite, thus the arrival rate doesn't depend upon the time. In our model we assume this to be true over each half hour period.

4.1.3 Arrival pattern

The ability of a queueing system to provide service for an arriving stream of calls depends not only upon the arrival rate, but also upon the pattern in which they arrive.

We assume call arrivals at times: $0 \leq t_0 < t_1 < t_2 < ... < t_n < ...$

The random variables $\tau_k = t_k - t_{k-1}$ are called inter-arrival times. $\tau_1, \tau_2, \tau_3, ...$ is assumed to be a sequence of independent identically distributed random variables. We therefore use the symbol τ for an arbitrary inter-arrival time. The arrival pattern most commonly assumed for queueing theory models is the exponential pattern: $P(\tau \leq t) = 1 - e^{-\lambda t}$, where λ is the average arrival rate. Because of the properties of the exponential distribution, if the inter-arrival time of calls to a queueing system has an exponential distribution, the arrival pattern is called a Poisson arrival pattern.

By assuming a Poisson arrival pattern, we consider the arrivals of calls as a sequence of independent events whose probability of occurrence is the same for any moment of some interval of time. Arrivals of telephone calls in a call centre do not strictly fall in this category but they often come exceedingly close to it. Therefore we will assume that they have both of the above properties, independence and constant probability in time.

As a consequence of these properties, i.e. since the calls arrivals are independent and have a constant probability in time, clusters of calls can occur. Therefore the number of agents had to be high enough to cope with clusters of calls, that is peaks in the call density. This situation is illustrated in Figure 4.1, which represents $P_x(t)$, the probability that x events have occurred by time t, as a function of x in a Poisson process. We assumed t to be equal to 10 seconds and λ, the arrival rate, to be equal to 5 arrivals per second. Then we computed $P_x(t)$ for different values of x. Of course the most likely case is that 50 calls have arrived by 30 seconds, but all other cases are possible and with a decreasing probability as the number of calls is far from 50. Hence 70 calls can arrived in 10 seconds, that is 40% more than the expected value, so the required number of agents has to deal with such clusters of call arrivals.

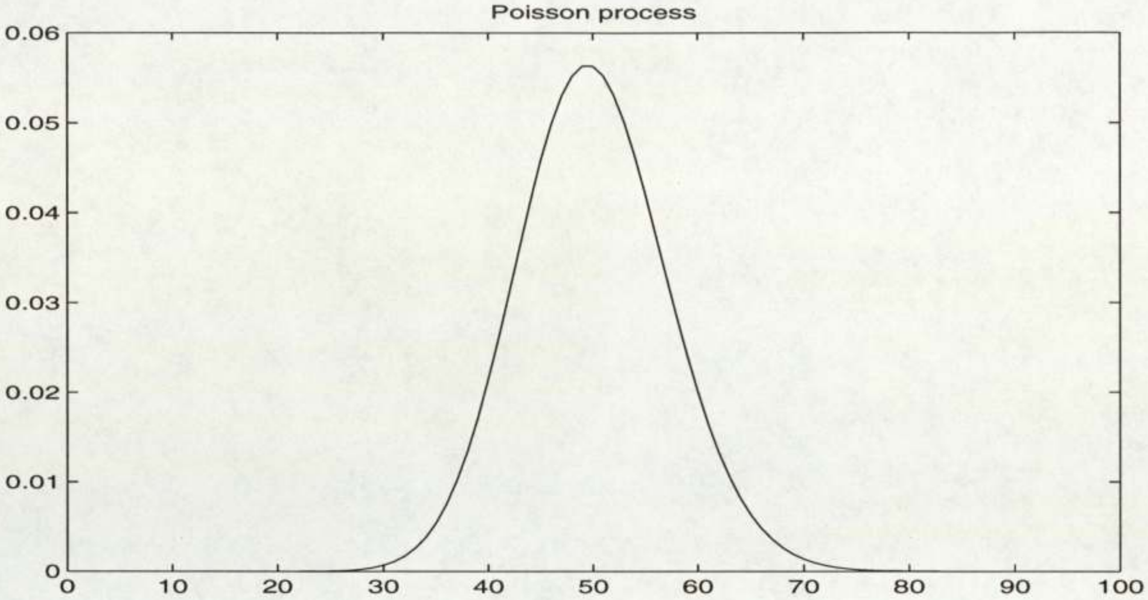


Figure 4.1: The Poisson process: $P_x(t) = f(x)$, the probability that x events have occurred by time t as a function of x. $t = 10$ seconds and $\lambda = 5$ arrivals/second.

4.1.4 Service time distribution

The exponential distribution is often used to describe the service time of a server. Thus if the service time is exponential, the expected time remaining to complete a call service is independent of the service already provided. In queueing theory, exponentially distributed service time is called random service and the distribution function $W_s(t)$ is given by:

$$W_s(t) = P[s \leq t] = 1 - e^{-\mu t}$$

It is difficult to give a sound theoretical reason why service times should be exponential; but the fact is that in most cases they are nearly exponential. It implies that the most likely duration of a call is around 0. It also means that the longer calls are the less likely to happen. This situation is depicted in Figure 4.2.

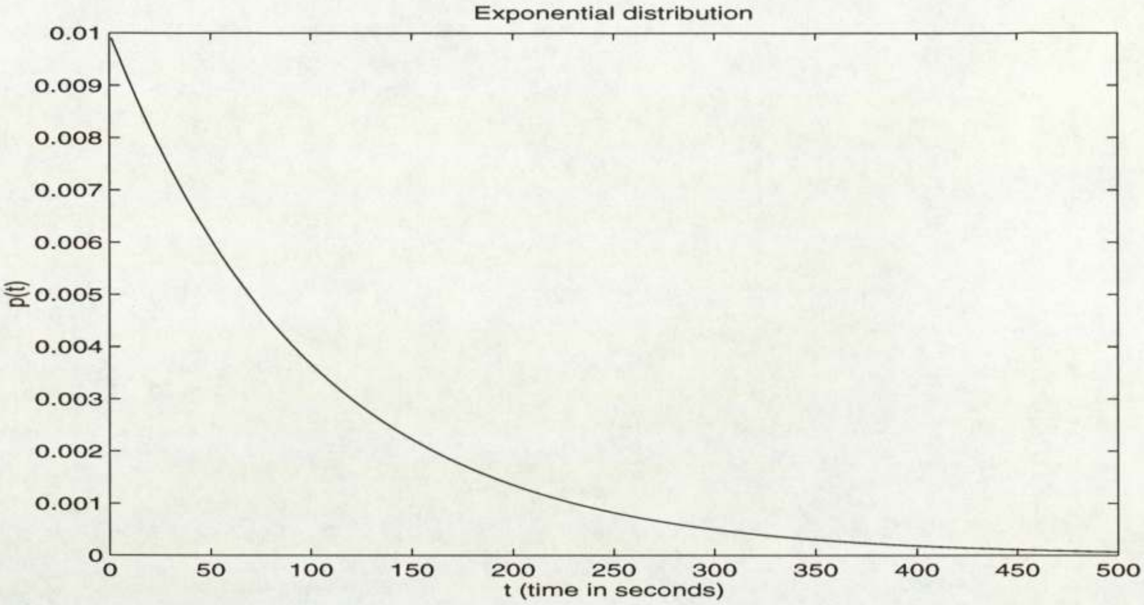


Figure 4.2: This figure illustrates an exponentially distributed service time. On the x axis, t represents the service time and on the y axis, $p(t)$ represents the probability that a call requires a service time of duration t.

4.1.5 Kendall notation

A shorthand notation, called the Kendall notation, after David Kendall, has been developed to describe queueing systems and has the form $A/B/c/K/m/Z$. Here A describes the inter-arrival time distribution, B the service time distribution, c the number of servers, K the system capacity (maximum number of customers allowed in the system), m the number in the source and Z the queueing discipline, that is the rule for selecting next customer to receive service. We will use the shorter notation $A/B/c$ as we assume that there is no limit to the queue size, the customer source is infinite and the queueing discipline is 'first come, first served'. The symbols we will use for A and B are:

- G: general service time distribution

- H_Q: Q-stage hyper-exponential service time distribution (it will be defined later)
- M: exponential inter-arrival or service time distribution

When we say a queueing model, such as $M/G/c$, has a general service time distribution, we mean the equations of the model are valid for very general service time distributions and, thus, in particular, the equations are valid for the $M/H_Q/c$ system.

4.2 A model of a call centre queueing system

In this section, we will present the current approach we use to describe the call centre queueing system and the mathematical notions required to model it. The assumptions made in the modelling of the system will also be discussed. For further details about all the mathematical notions used in this section, see appendix B.

4.2.1 Introduction

We consider a set of Q queues, each of which contains calls of only one type, that is to say with one arrival rate and one service rate. The arrival pattern used to described the arrival of the calls in the different queues is a Poisson process, with arrival rate λ_q depending upon the queue. We assume that the service time for each type of calls is exponentially distributed with service rate μ_q depending upon the queue. The description of a Poisson process and the meaning of an exponential distribution are discussed in more detail in appendix B .

We have seen in the previous section that in queueing theory, it is usual to consider a single queue and a single service time distribution. Each type of queueing system is characterised by an inter-arrival time distribution, a service time distribution and a number of servers. To model our system, we need to find an inter-arrival time distribution and a service time distribution which fit our multiple queues system. That is what is discussed in the following section.

In queueing theory, there are some formulas which give the service level reached with a queueing model. Thanks to such formulas we can know how many servers we need in this system to achieve a certain service level. On the test problem, the desired service level is the following: 95% of all the calls should be answered within 30 seconds. Once we know the required total number of agents in the system, we have to share them between different skills mix pools thanks to the optimisation of the quality ratio, which was discussed earlier. The quality ratio takes into account the differences between the skills required by the incoming calls and the real skills of the agents who have to answer the calls. By optimising it, we can find the right distribution of the agents between the skills mixes.

4.2.2 Exponential inter-arrival time

In each queue, the inter-arrival time distribution is assumed to be exponential. Because of the properties of the exponential distribution, summarised in appendix B, the arrival pattern is a Poisson arrival pattern. For each queue q, the arrival rate λ_q is computed using the call volume, i.e. the number of incoming calls in an half hour period, denoted N_q, and is assumed to be constant throughout an half hour period:

$$\lambda_q = \frac{N_q}{30 \times 60}$$

In our approach, all the agents can answer calls from any queue and so the different queues are merged in one system. So it is relevant to model the arrivals of the calls from different queues in our system as one process, as we are trying to do.

The sum of independent Poisson processes is itself a Poisson process, whose parameter is the sum of the parameters of the different Poisson processes. As each of the types of call has an independent Poisson arrival pattern with parameter λ_q, the single queue to our system has also an Poisson arrival pattern, whose arrival rate parameter λ is equal to:

$$\lambda = \sum_{q=1}^{Q} \lambda_q$$

It is easy to understand how to derive the new arrival rate. As the arrival rate of each queue is equal to the number of incoming calls during an half hour period divided by the number of seconds in an half hour period, i.e. $\lambda_q = N_q/(30 \times 60)$, the global arrival rate is computed with the total number of calls from every queue:

$$\lambda = \frac{\sum_{q=1}^{Q} N_q}{30 \times 60} = \sum_{q=1}^{Q} \left(\frac{N_q}{30 \times 60} \right) = \sum_{q=1}^{Q} \lambda_q$$

To conclude, we can use an exponential distribution to model the inter-arrival time of every type of incoming call.

4.2.3 Hyper-exponentially distributed service time

In our model each agent answers different types of incoming call. The service time distribution for each type of call is exponentially distributed and with a different parameter for each queue. However we considered that all the queues are merged in one, so the service time distribution has to be modelled according to a combination of the service time distributions of each type of call. This can be done using a Q-stage hyper-exponential distribution, which is used for modelling the service time of a queueing system, which has a large standard deviation relative to the mean value.

For each queue q, the service rate μ_q is computed using the mean duration of a call in an half hour period, which is denoted T_q, and is assumed to be constant in an half hour period:

$$\mu_q = \frac{1}{T_q}$$

As there are Q types of call, so Q different mean call duration values, we will use a Q-stage hyper-exponential distribution to model our service time. The density function of this distribution is equal to:

$$f_s(t) = \sum_{q=1}^{Q} \alpha_q \mu_q e^{-\mu_q t} \quad \text{with} \quad \alpha_q = \frac{\lambda_q}{\sum_{q=1}^{Q} \lambda_q} \tag{4.1}$$

Each server is considered as a Q-stage hyper-exponential service facility, that means each agent has the ability to provide Q different services, each with an exponentially distributed service time. This situation is depicted on Figure 4.3.

Each α_q represents the probability that an agent answers a call of type q. Thus the probability that an incoming call is one of type q is equal to $\alpha_q = \frac{\lambda_q}{\sum_{q=1}^{Q} \lambda_q}$.

An important point concerning the formula 4.1 describing the service time distribution is that we are using a stochastic model of the service time, which is usual, and call types, which is less usual. Indeed, the coefficients α_q enable us to take into account the relative importance of each type of call in the system.

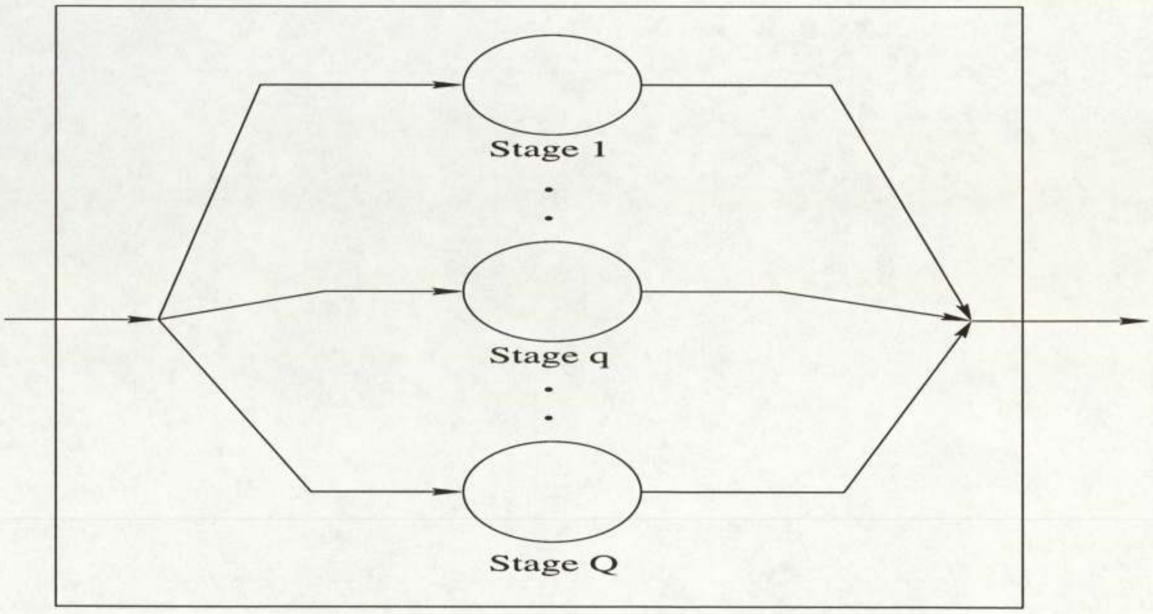


Figure 4.3: Model to represent a Q-stage hyper-exponential service facility.

4.2.4 Conclusions

We consider a system of Q queues, each of which has exponential inter-arrival and service time distributions. The parameters used to describe this Q queues are denoted λ_q for the arrival rates and μ_q for the service rates. Our current queueing system can be described by an $M/H_Q/c$ model:

- arrival pattern: exponentially distributed inter-arrival time with an arrival rate given by $\lambda = \sum_{q=1}^{Q} \lambda_q$
- the service time distribution is hyper-exponential: the density function is equal to $f_s(t) = \sum_{q=1}^{Q} \alpha_q \mu_q e^{-\mu_q t}$, with $\alpha_q = \frac{\lambda_q}{\sum_{q=1}^{Q} \lambda_q}$

Once we have found a relevant model for our system, we need to derive the number of servers required to achieve a satisfactory service level. This will done in the next section.

4.3 Required number of servers

In this section, we will present the mathematical notions and formulas needed to derive the number of servers required in our system. We will also describe how it can be derived.

4.3.1 Introduction

In the previous section, we have discussed the model used to represent our system. The model found is an $M/H_Q/c$ model, that is a queueing model with exponential inter-arrival time distribution, Q-stage hyper-exponential service time distribution and c servers, i.e. c agents. The purpose of this section is to describe how to derive the number of agents required in such a model to achieve a sufficiently high service level. Therefore we will introduce some mathematical notations and formulas and make some assumptions.

4.3.2 Martin's estimate

We shall consider an M/G/c queueing system. Such a system has an exponential inter-arrival time distribution, a general service time distribution and c servers. As detailed earlier in section 1, d denotes the random variable describing the time spent in queue, and the expected value of the time spent in queue is denoted by $W_d = E[d]$. So W_d can be considered as the average waiting in queue and, for this system, one can approximate the average waiting time in queue, $W_d = E[d]$, by:

$$W_d = \frac{C(c,a) \times E[s]}{c-a} \times \frac{1+C_s^2}{2} \qquad \text{Martin's estimate} \qquad (4.2)$$

where:

- s is the random variable describing the service time
- $E[s]$ is the expected service time
- $a = \lambda \times E[s]$ and λ is the arrival rate
- $C(c,a)$ is the Erlang's C function (see appendix B)
- $C_s^2 = \frac{Var[s]}{E[s]^2}$ is called the squared coefficient of variation of s

Formula 4.2 is called Martin's estimate, [2], and, as it is not an exact formula but an estimate, it might not be entirely true. However the results are quite accurate and the error is nearly zero. Such an estimate has the important advantage of being relatively easy to calculate. To find an exact formula for the model considered, i.e. an $M/H_Q/c$ model, would have been excessively long and the formula obtained would have been probably difficult to calculate and computationally expensive.

An important point is that in our system the only unknown is the number of servers and consequently W_d depends only upon it. So the average waiting time in queue can be considered as a function of c, the number of agents in the system.

4.3.3 Heavy Traffic Approximation

We shall consider a G/G/c queueing system, i.e. general inter-arrival and service time distributions and c servers. An important variable in queueing theory is ρ, the server utilisation, which measures the average utilisation of a server. It is equal to $\lambda/(c \times \mu)$, that is the ratio of the number of arriving calls per unit of time and the number of calls per unit of time that the system is able to complete. It is relatively straight forward to see how ρ can be used to measure the importance of the traffic in a queueing system:

- If ρ approaches 1, then the traffic can be considered as heavy
- If ρ approaches 0, then the traffic can be considered as light

As ρ approaches 1, the distribution of the queueing time, d, approaches that of an exponential distribution, see section 4.1.4. That is the heavy traffic approximation, which suits our case well as we want to minimise the number of staff, which implies a server utilisation close to 1. To consider the distribution of our queueing time close to an exponential distribution allows us to compute easily the percentile value of d, the random variable describing the queueing time.

The r^{th} percentile value of d, denoted by $\pi_d(r)$, can be used as a measure for the service level. For instance, if $\pi_d(90)$, the 90^{th} percentile value of queueing time, is equal to 40 seconds, then 90 percent of all calls spend less than 40 seconds in the queue. So the r^{th} percentile value of d, $\pi_d(r)$, is defined by:

$$P(d \leq \pi_d(r)) = \frac{r}{100}$$

Assuming the heavy traffic approximation, and consequently that the distribution of the queueing time, d, approaches that of an exponential distribution, one can use the following formula to calculate the r^{th} percentile value of d:

$$\pi_d(r) = E[d] \times \ln\left(\frac{100}{100-r}\right) = W_d \times \ln\left(\frac{100}{100-r}\right)$$

The justification of this formula is given in appendix C.

This relation between the r^{th} percentile value of q and the expected queueing time gives the required number of agents to achieve our service level, as the expected queueing time W_d can be considered as a function of c, the number of servers.

4.3.4 Required number of servers

Once we have decided on the model to describe our system, we can derive the required number of servers in our system, thanks to the formulas and approximations made earlier.

In our example, the service level will be achieved if 95% of the incoming calls are answered within 30 seconds. Therefore the 95^{th} percentile value, $\pi_d(95)$, must be equal at least to 30 seconds, which is equivalent to $P(d \leq 30) = 0.95$. Using the heavy traffic approximation, $\pi_d(95) = W_d \times \ln(\frac{100}{5}) \approx 3 \times W_d$. So $\pi_d(95) \approx 3 \times W_d$ must be equal or less than 30 seconds, which means that we have to find the minimum number of servers such that W_d is equal or less than 10 seconds.

Thanks to Martin's estimate, the expected waiting time can be easily computed in the case of an $M/H_Q/c$ system:

$$W_d = \frac{C(c,a) \times E[s]}{c-a} \times \frac{1+C_s^2}{2}$$

And, as the service time distribution is hyper-exponential, the different terms of this formula can be computed:

- $E[s] = \sum_{q=1}^{Q} \frac{\alpha_q}{\mu_q}$
- $E[s^2] = 2\sum_{q=1}^{Q} \frac{\alpha_q}{\mu_q^2}$
- $C_s^2 = \frac{Var[s]}{E[s]^2} = \frac{E[s^2]}{E[s]^2} - 1$

- $C(c, a)$, the Erlang's C function, can be derived thanks to an iterative formula

More detail about this calculations can be found in appendix C.

Since c is the only unknown in this formula, we can consider W_d as a function of c. So we can derive the required number of servers by finding the value of c for which W_d matches the right value of the waiting time. That means that the required number of servers will be the smallest value of c for which W_d will be less than 10 seconds.

4.4 Results

4.4.1 The cost function

Using the method described in the previous section, we can compute the required number of servers, i.e. agents in the system, for each half hour period throughout the whole week. So we will have to work now with the required number of staff, computed with the queueing model, and not anymore with the number of active calls. This can be done because the sum $\sum_{p=1}^{P} c_{pq}$, which formerly was seen as the number of agents to allocate to all the calls active in queue q at an instant, can now be regarded as the number of agents required to answer calls from queue q. Therefore this sum, $\sum_{p=1}^{P} c_{pq}$, now has to match n_q, the predicted number of agents required to answer the calls from queue q. The meaning of the cost function will then slightly change. This number will be derived thanks to c, the total number of servers in the system. We consider that n_q is equal to c times the ratio of active calls in the queue q and the total number of calls active in the half hour period:

$$n_q = \frac{N_q T_q}{\sum_{q=1}^{Q} N_q T_q} \times c$$

where N_q is the number of incoming calls in queue q during an half hour period and τ_q is the average call duration in queue q.

Another important point, in this new approach, is that we require a minimum number of servers in the system for each half hour period. Thus we have to model this constraint, according to the fact that if the total number of agents in the system is less than the minimal one, the service level won't be reached. It is then necessary to use a hard constraint to model this, such as the penalty function method. If the constraint is violated, we add to the cost function a penalty term:

$$\text{If} \quad \sum_{q=1}^{Q}\sum_{p=1}^{P} c_{pq} < c, \quad \text{then} \quad E_{new} = E_{old} + \frac{\delta}{2}\left(\sum_{q=1}^{Q}\sum_{p=1}^{P} c_{pq} - c\right)^2$$

The cost function that we have mentioned is the same that we have used before. It always contains the following terms:

- a term measuring a loss in quality:

$$E_0 = \frac{1}{Q}\sum_{q=1}^{Q}\left(\sum_{p=1}^{P} c_{pq}S_{pq} - n_q\right)^2$$

- a term constraining the sum on each queue to equal the number of calls:

$$E_1 = \frac{1}{Q}\sum_{q=1}^{Q}\left(\sum_{p=1}^{P} c_{pq} - n_q\right)^2$$

- a term corresponding to the minimisation of the number of staff:

$$E_2 = \frac{1}{Q}\sum_{q=1}^{Q}\sum_{p=1}^{P} c_{pq}^2$$

The final cost function we are interested in is obtained by combining all the above terms as follows:

$$E = E_0 + \alpha E_1 + \beta E_2$$

where α and β are Lagrange multipliers which may be set to specific values which reflect our relative importance of satisfying each of the penalty terms.

And we need the following constraints to be implemented using the penalty function method:

- no connectivity should be negative:

$$\text{If} \quad c_{pq} < 0, \quad \text{then} \quad E_{new} = E_{old} + \frac{\delta}{2}c_{pq}^2$$

- no call should be answered by people completely unqualified:

$$\text{If} \quad S_{pq} = 0 \quad \text{and} \quad c_{pq} > 0, \quad \text{then} \quad E_{new} = E_{old} + \frac{\delta}{2} c_{pq}^2$$

- the number of agents in the system should be greater than c, that is the minimum number of agents required to achieve the right service level:

$$\text{If} \quad \sum_{q=1}^{Q} \sum_{p=1}^{P} c_{pq} < c, \quad \text{then} \quad E_{new} = E_{old} + \frac{\delta}{2} \left(\sum_{q=1}^{Q} \sum_{p=1}^{P} c_{pq} - c \right)^2$$

An optional constraint can of course be added, if it is necessary, to cap the number of people in some skills mixes. We should then consider the following penalty term:

$$\text{If} \quad \sum_{q=1}^{Q} c_{pq} > A_p, \quad \text{then} \quad E_{new} = E_{old} + \frac{\delta}{2} \left(\sum_{q=1}^{Q} c_{pq} - A_p \right)^2$$

4.4.2 Results

As expected the number of agents required increases compared to the one found with uniformly distributed calls, so that the workforce can cope with the fluctuations in the call density. This situation is depicted on Figure 4.4. The increase in the number of agents is around 5%, compared to the simple model used in the feasibility study. The total reduction in the workforce remains quite high, that is 20% fewer agents required in the whole week, as is shown on Figure 4.5.

Now we have to consider two stages before obtaining the final distribution of the agents between the pools. First we have to run an algorithm which computes the total number of agents required in the system for each half hour period in order to achieve a satisfactory service level. This first stage is quite fast, it takes barely a few minutes with a Matlab program. Then we use the number of agents required in the system to run our optimisation algorithm and to share the agents between the different pools. This stage uses the same algorithm that the one used before in the simple model, in which we have added the constraint which prevents the total number of agents in the system from being less than the one required to achieve the service level. So it takes a little bit more than 10 minutes to run the algorithm through the data of a whole week.

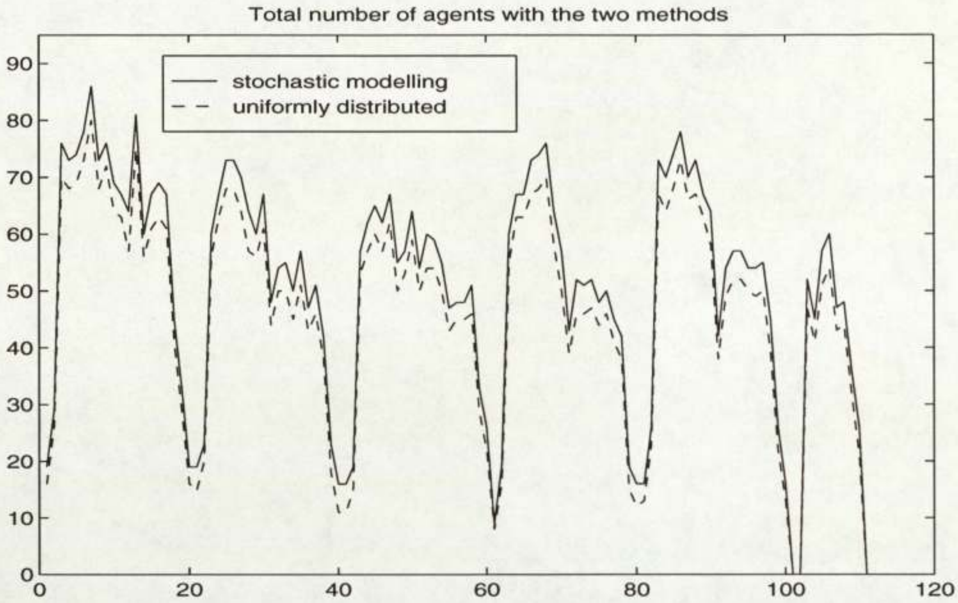


Figure 4.4: Comparison between the number of agents required with uniformly distributed calls (the dotted line) and with the stochastic model of the queues (solid line).

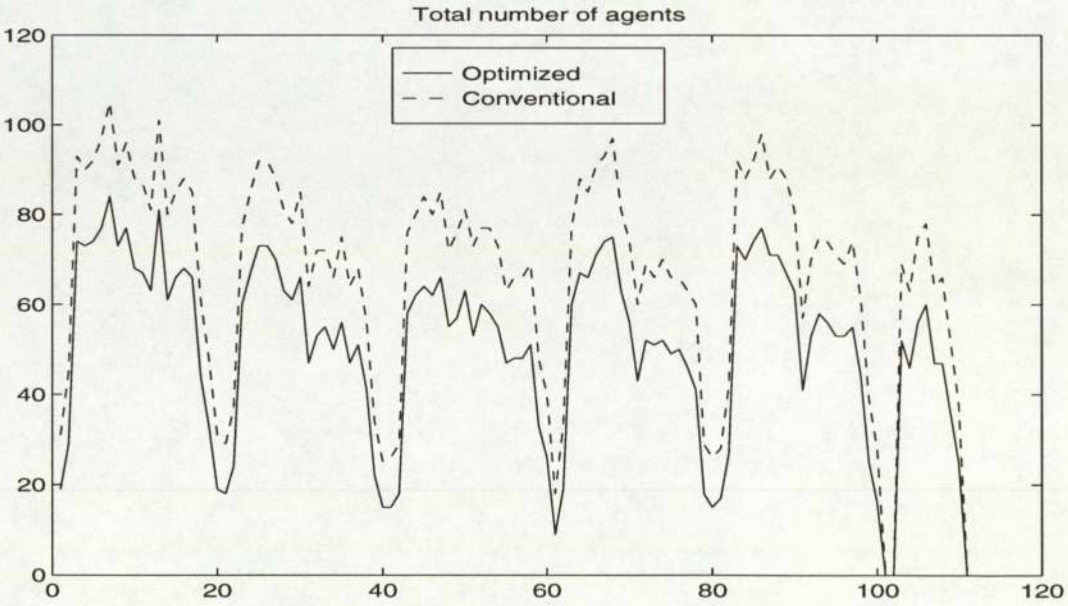


Figure 4.5: Comparison between the number of agents used in the traditional model (the dotted line) and the number of agents used in the stochastic model of the queues (solid line).

4.5 Conclusions

The results found by using a stochastic model of the queues are quite close to the ones found using the assumption that the calls are uniformly distributed. This can be explained by the fact that by spreading the queues across a number of skills mix groups it is easier to cope with fluctuations in call arrivals. Another reason in the small importance of the increase lies in the delay of 30 seconds that we now allow to the agents before answering a phone call. In the previous approach, the agents were allocated on a instantaneous basis to active phone calls. Now we consider the waiting time in the queue, which smooth the fluctuations. It is also worth noting that we don't take into account the calls that are given up when the delay is too long before they are being answered. This naturally increases the number of agents required as we assume that all the incoming calls have to be answered, which is not the case in reality where some calls are abandoned.

The next step is to run some simulations in order to validate our model. We have made in this study some assumptions and approximations that needed to be tested on a real simulation:

1. Martin's estimate to model an M/G/C queueing system may not be as accurate as expected;
2. the Heavy Traffic Assumption, which seemed to be valid for our system, as we consider a minimal workforce, needs also to be validated through a simulation;
3. the hyper-exponentially distributed service time may be a too simple way of modelling the actual service time.

Chapter 5

Validation

In this chapter, we present the different methods we have used to validate our approach. One of them is to observe and analyse the results obtained with different data sets.

First we have built new data sets by scaling up the one that we have used so far. The two kinds of data that we consider are the call volume, i.e. the number of incoming calls to the system, and the mean call duration. By observing how changes the number of agents required when we increase these two parameters, we could derive some rules on the behaviour of our approach against the size of the data.

Secondly we have run our algorithm through new data sets provided by CallScan. By comparing the number of agents required with our method and with the traditional one, we could conclude on whether the reduction obtained in the number of agents is strongly linked to the nature of the data set.

Another way to validate our approach is to run an algorithm that simulates the activity of a call centre for a whole week. Such simulation is based on some assumptions that are commonly used to model the behaviour of the queues of incoming calls in a call centre.

5.1 New data sets

5.1.1 Scaling-up the data set

On the basis of the first data set provided by CallScan, we created new data sets by:

- Increasing the call volume
- Increasing the average call duration
- Increasing the call volume and the average call duration

For each of this three cases, we computed the number of agents required in our new approach, using the stochastic modelling of the queues, alongside the number of agents predicted in the traditional approach, i.e. using the Erlang's C model. This two numbers are compared using figures showing the numbers of agents required in both methods throughout the whole week.

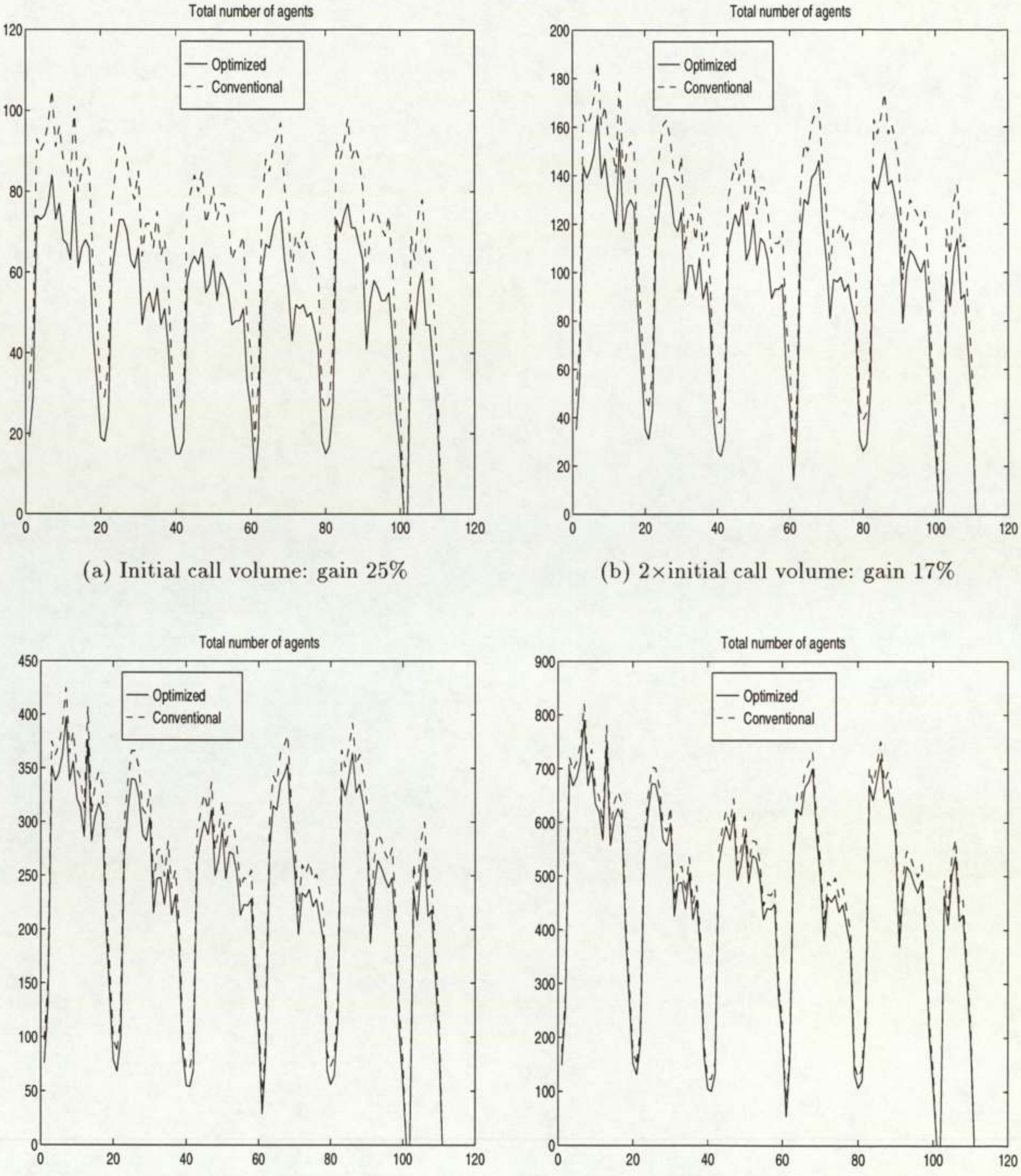
In figure 5.1, the number of agents required in both methods is presented for the initial call volume and for the initial call volume increased by 2, 5 and 10. We obtain four plots, each corresponding to a different scale of the initial data set. By increasing the call volume, we increase the arrival rate, hence the number of agents will increase but differently with the two methods. The number of agents required in our model increases more than the one required with Erlang's C model. Therefore the percentage of the number of agents gained with our new approach compared with the traditional number of agents, i.e. the one predicted in Erlang's C model, will decrease. Actually it decreases from 25% with the initial data set to 6% with a call volume increased by 10. This reduction in the staff may decrease as a percentage but still increases in term of number of agents. For instance, the average reduction in the number of agents with our method compared to the traditional one for an half hour period is 17 agents with the initial data set, 20 with a call volume increased by 2, 23 with a call volume increased by 5 and 28 with a call volume increased by 10.

In figure 5.2, the number of agents required in both methods is presented for the initial mean call duration and for the initial mean call duration increased by 2, 4 and 6. We obtain four plots, each corresponding to a different data set. By increasing the mean call duration, we decrease the service rate and therefore the number of agents will increase but differently with the two methods. The situation is the same than the previous one, i.e. when we increased the call volume. The reduction of staff with our new method compared to the traditional one is decreasing as a percentage, but not in the number of agents, as is shown on the figure 5.2.

In figure 5.3, the number of agents required in both methods is presented for the initial data set and for the initial call volume and mean call duration respectively increased by 2 and 2, 5 and 4 and 10 and 6. We obtain four plots, each corresponding to a different data set. By increasing the call volume and the mean call duration, we both increase the arrival rate and decrease the service rate, therefore the number of agents will increase but differently with the two methods. The situation is the same than the previous ones, i.e. when we increased the call volume or when we increased the mean call duration. The gain in agents with our new method compared to the traditional one is decreasing as a percentage, but not in the number of agents. as is shown on the figure 5.3.

The decrease in the percentage of agents gained with our method corresponds to the decrease in the proportion of the clustering as the call volume and the call duration increase. The clustering concerning the arrivals means an increase in the call density, i.e. more calls in the same amount of time. As for call duration, the clustering means more calls requiring long service time incoming together in the call centre. In both cases, the proportion of the clustering decreases as these parameters increase. Since our approach copes well with such clustering by spreading the handling of queues across a number of skills mix groups (and a potentially larger pool of people for any given queue), its advantage over the traditional model becomes less important as the activity of the call centre increases and the proportion of clustering decreases.

Increasing the call volume



(a) Initial call volume: gain 25%

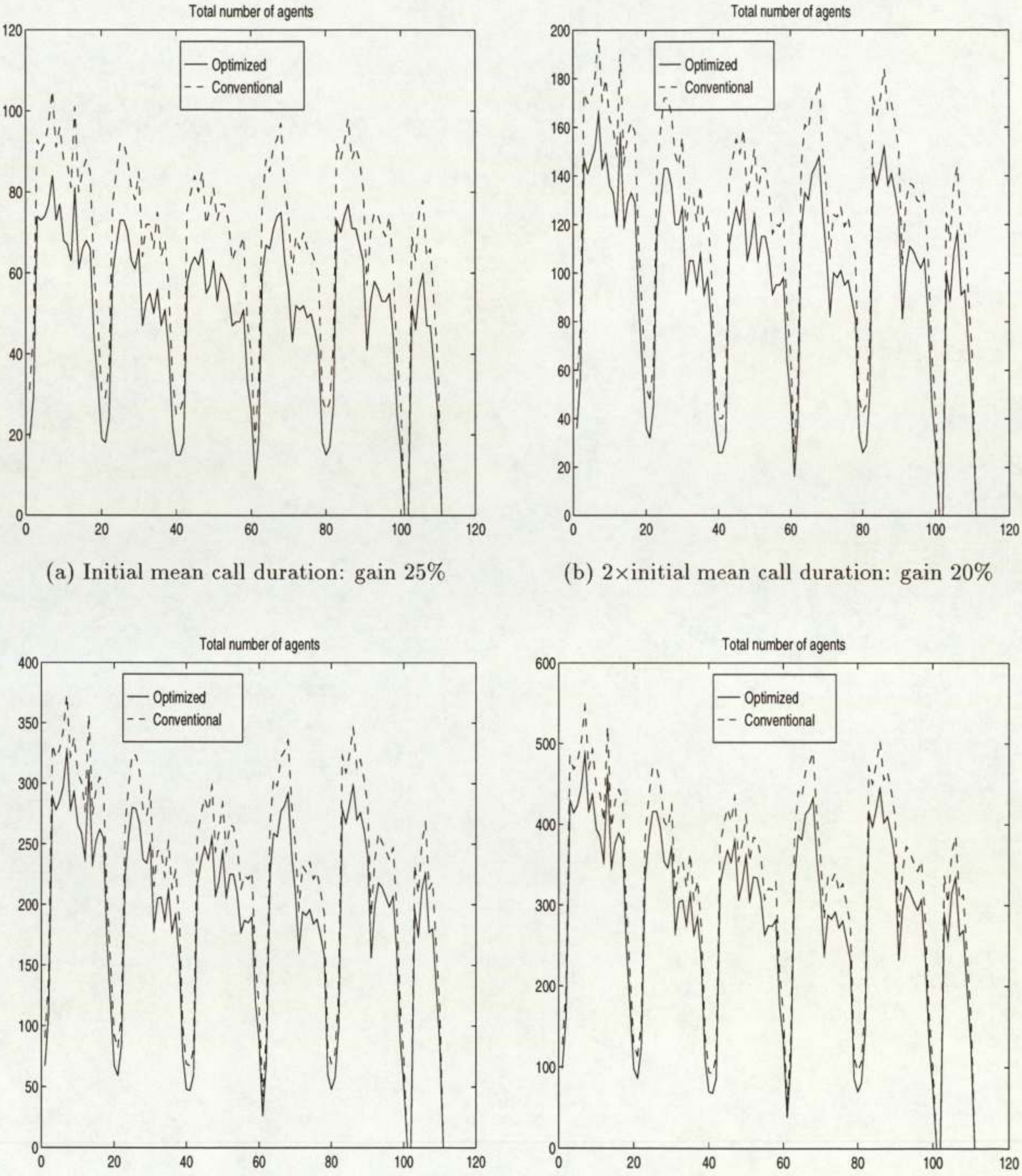
(b) 2×initial call volume: gain 17%

(c) 5×initial call volume: gain 10%

(d) 10×initial call volume: gain 6%

Figure 5.1: Comparison between the number of agents used in the traditional model (the dotted line) and the number of agents used in our approach (solid line) when increasing the call volume.

Increasing the mean call duration



(a) Initial mean call duration: gain 25%

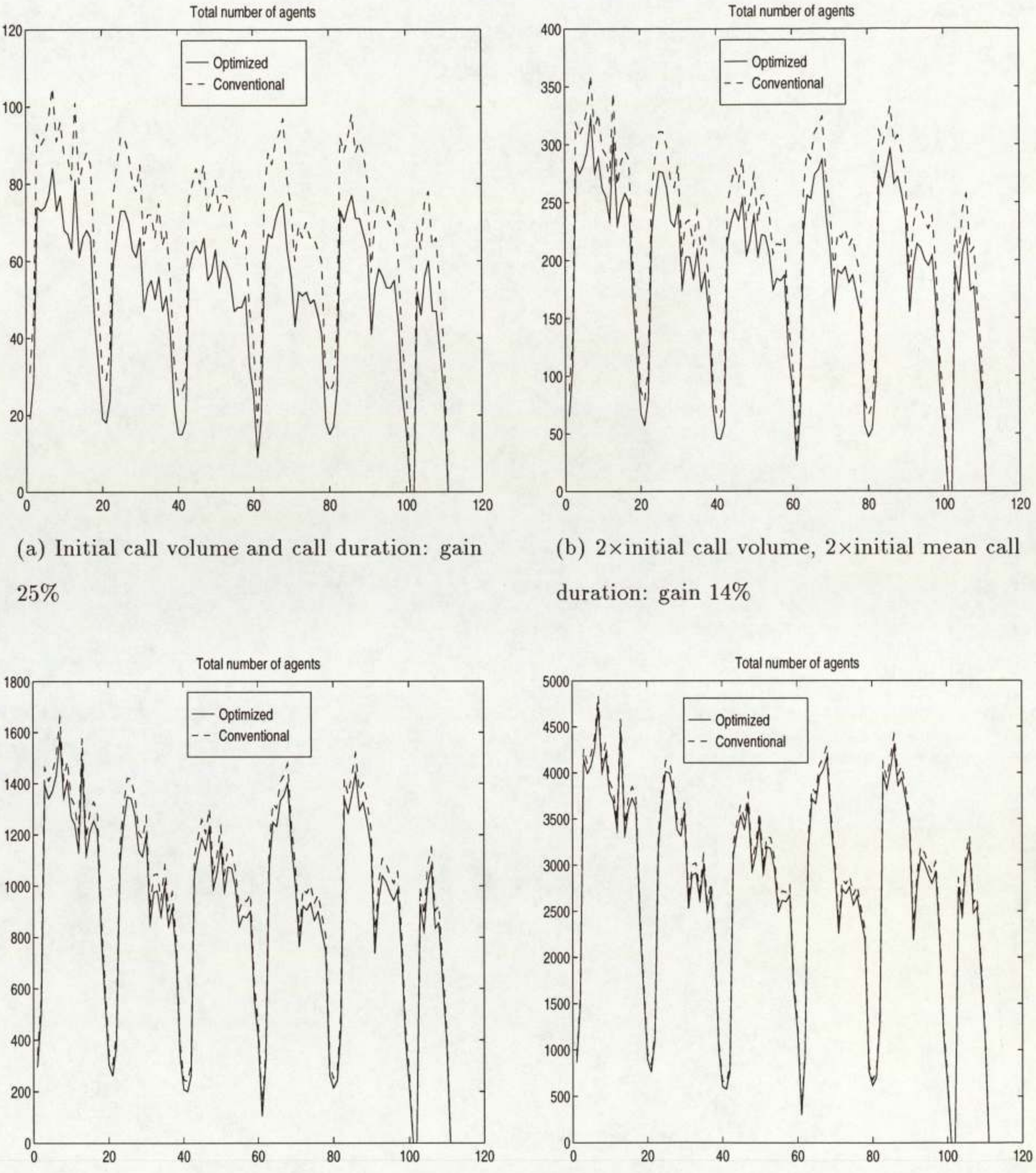
(b) 2×initial mean call duration: gain 20%

(c) 4×initial mean call duration: gain 16%

(d) 6×initial mean call duration: gain 14%

Figure 5.2: Comparison between the number of agents used in the traditional model (the dotted line) and the number of agents used in our approach (solid line) when increasing the mean call duration.

Increasing the call volume and the mean call duration



(a) Initial call volume and call duration: gain 25%

(b) 2×initial call volume, 2×initial mean call duration: gain 14%

(c) 5×initial call volume, 4×initial mean call duration: gain 7%

(d) 10×initial call volume, 6×initial mean call duration: gain 4%

Figure 5.3: Comparison between the number of agents used in the traditional model (the dotted line) and the number of agents used in our approach (solid line) when increasing the call volume and the mean call duration.

5.1.2 Tests with different data sets

Two new data sets have been provided by CallScan in order to validate our approach. These data sets contain three types of data:

- the call volume, i.e. the number of incoming calls
- the mean call duration
- the number of agents in the call centre

These data are collected from a call centre for a whole week of work. They described the activity of the call centre for every half hour period of the week, i.e. from Monday to Sunday and from 7:30 to 23:00. So we consider 217 half hour periods and for each of them we compute the number of agents predicted with our method. The actual number of agents in the call centre differs slightly from the number of agents required with a Erlang's C model using the two first parameters. This is explained by the fact that the actual number of agents in the call centre is computed with an Erlang's C model but with predicted rather than actual values of call volume and mean call duration. For a fair comparison between our model and the Erlang's C model, we should compare our results with the results calculated with Erlang's C model for the actual data.

Since the two data sets provided by CallScan describe a single queue, we consider that six identical queues of incoming calls arrived to the call centre. We also compute the measure of quality of service provided to the calls. This measure will tell us how well the calls are answered. This measure is a percentage which will take on the value 1 if all the calls are answered by people totally qualified.

First data set

This data set comes from a very busy call centre, as the number of agents required is very high. The service level considered in this example is 85% of all the calls answered within 20 seconds. As we are considering a call centre with a high level of activity, we observe a quite small percentage reduction in the number of agents required in our method compared to the traditional one: we need 5% fewer agents in our approach. However, since the number of agents required in both models is quite high, a 5% reduction means 45 agents less in average for each half hour period. The number of staff required for this first data set is shown in Figure 5.4, and the evolution of the quality ratio throughout the whole week in Figure 5.5.

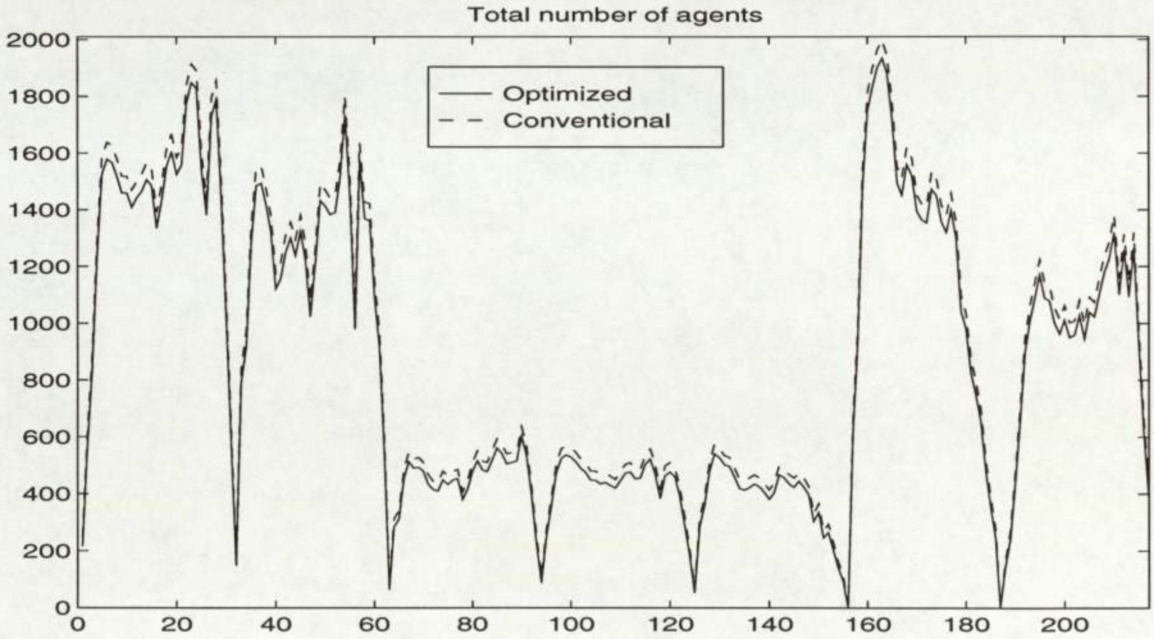


Figure 5.4: Comparison between the number of agents used in the traditional model (the dotted line) and the number of agents used in our approach (solid line) with the first new data set.

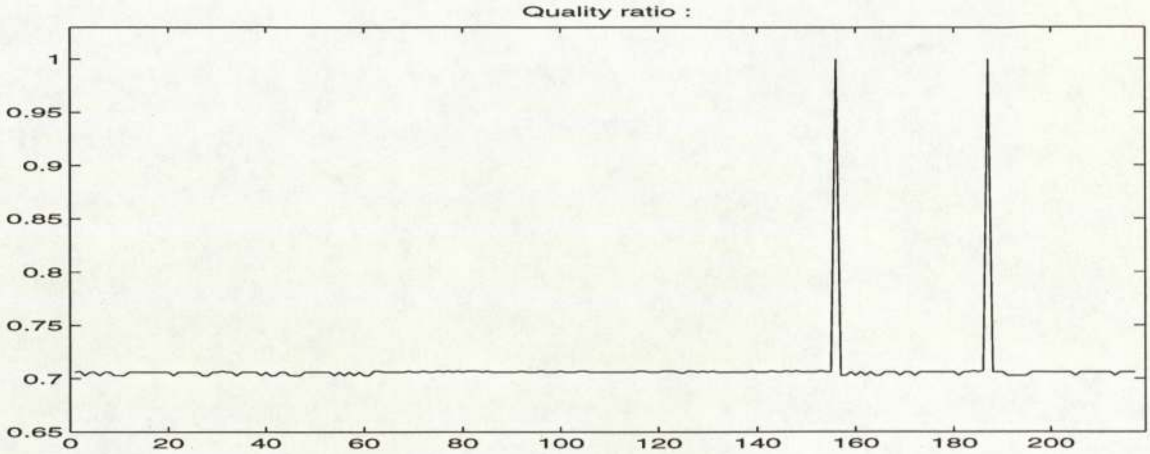


Figure 5.5: The quantification of the quality of service provided by the skills mix approach method with the first new data set. On this scale a value of 1 indicates perfect quality.

Second data set

This data set represents a call centre with a reasonably high activity. The percentage of agents gained with our method is around 10%, which is what we expected according to the tests made with the scaled-up data set. The number of staff required for this first data set is shown in Figure 5.6, and the evolution of the quality ratio throughout the whole week in Figure 5.7.

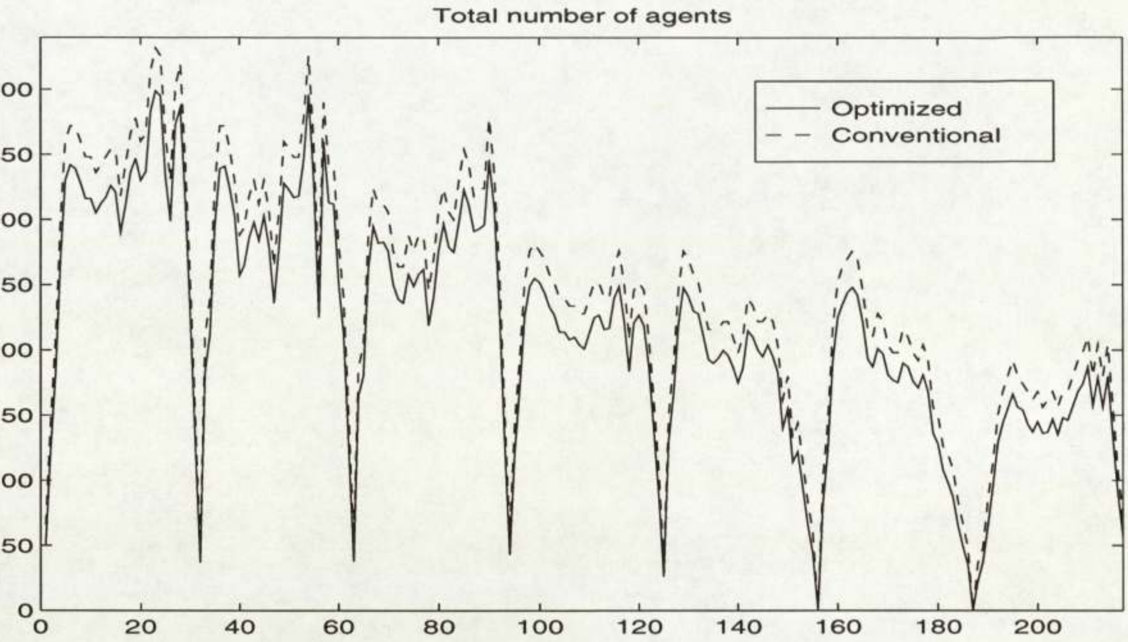


Figure 5.6: Comparison between the number of agents used in the traditional model (the dotted line) and the number of agents used in our approach (solid line) with the second new data set.

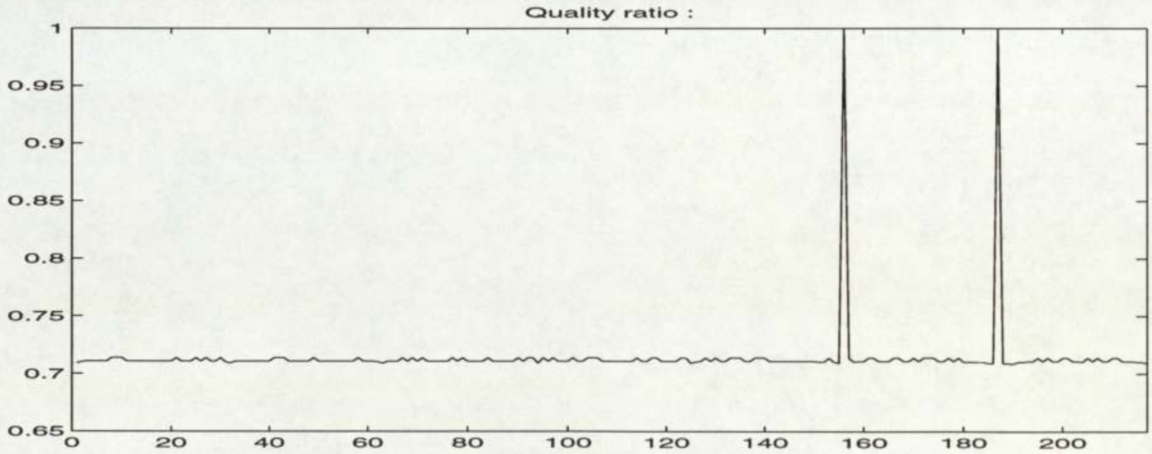


Figure 5.7: The quantification of the quality of service provided by the skills mix approach method with the second new data set. On this scale a value of 1 indicates perfect quality.

5.2 Simulation

In this section, we discuss how we validated our new approach through simulation. Therefore we first have to justify and explain the algorithm used to simulate the activity of a call centre. This simulation aims to validate our approach by observing how a realistic model of a call centre behaves. The measures we will use to estimate how well our approach works, are the service level, i.e. the percentage of calls answered within the maximum acceptable delay, and the quality of service provided to customers.

5.2.1 Assumptions

In the simulation, we will use the initial data set provided by CallScan to simulate the behaviour of the queues. That means we will consider six queues of incoming calls, six working days, each with ten hours of work and consequently 120 half hour periods of work. For each half hour period and each queue, we know the call volume, i.e. the number of incoming calls, the mean call duration and the predicted number of agents required to answer the calls. The predicted number of agents we are considering is the one given by our approach and with a stochastic modelling of the queues.

To model the activity of a call centre, we have made the following assumptions:

- Poisson arrivals
- Exponentially distributed service time
- No agent is allowed to answer calls for which he has 0% competency
- An incoming call is allocated to the most qualified agent available
- The longest waiting call is always the one selected for service
- The customers are assumed to be infinitely patient

We consider that the service level to achieve is 95% of calls answered in 30 seconds. In our simulation we used the skills matrix in table 2.1.

5.2.2 Simulation

According to the Poisson process, either zero or one calls can arrive in any small time interval, as explained in more detail in appendix B. So we work with the second as the time basis, as it could be consider as a small time interval: in a second either zero or one call arrive in each of the six queues. Thus, in our algorithm, each second of each half hour period is considered as a step. Here are the different actions done by our algorithm at each step, that is for each second of a given half hour period:

- In each queue either one or zero calls arrive, i.e. between 0 and 6 calls arrive in our system;
- We decrease the remaining service time of each agent answering a call by one second;
- An agent is available if his remaining service time is equal to zero;
- The longest waiting call is allocated to the most qualified agent available;
- The time required to answer the call is taken randomly from an exponential distribution;
- We increase by one the waiting time of the calls still waiting in one of the queues.

The random process used to model the call arrivals and the service time is describe in appendix D. For each half hour period in the week, the fraction of calls that have experienced a waiting time greater than 30 seconds is computed. The quality of service provided to the customers is also computed using the skills score of the agents answering the phone calls: the quality of service is the average competence with which the calls are being answered by the agents. This measure of quality reaches the value 1 if all the calls have been answered by agents with 100% competencies. Of course this situation won't happen because, one of the advantages of our approach is that we take into account all the skills of every agent, so that an agent can answer calls for which is not completely qualified. However we are striving to maximise our measure of quality.

5.3 Results of the simulation

5.3.1 Initial results

These are the first results given by our simulation program. These results are the average of several simulations, each run with a different random seed [9]. These results are quite acceptable, as the service level is often greater than 95% of calls answered within 30 seconds. However the service level is not reached in 17 half hour periods, amongst the 120 of the whole week, as shown in Figure 5.8. This happens mainly for half hour periods with a small number of calls, as it will be detailed later. Consequently if we consider the service level for the whole week instead of the service level for every half hour period, we obtain the following result: only 2.75% of calls are not answered within 30 seconds. So globally, the service level achieved is quite good but locally, in some half hour periods, it is not satisfactory. As for the quality of service provided to the calls, the results are better than the ones expected, as the average competence of an agent answering a phone call is 81%, as shown in Figure 5.9.

The relative weakness of the results concerning the service level has at least three causes. First, in our model, we have assumed that any agent can answer any type of call. This is not true since we have decided that an agent should answer only the calls for which he has some competencies, i.e. a skills score non null. Secondly, an important point is the sharing of the agents between the queues. In our new approach, we use queueing theory to determine the total number of agents in our system and then we use a cost function to derive how this number of agents has to be shared between the different skills mixes.

To illustrate the importance of this point, we can consider a pool whose agents are more qualified than the ones of the others. As we allocate the incoming calls to the most qualified agent available, the agents of this pool are the first to be answering

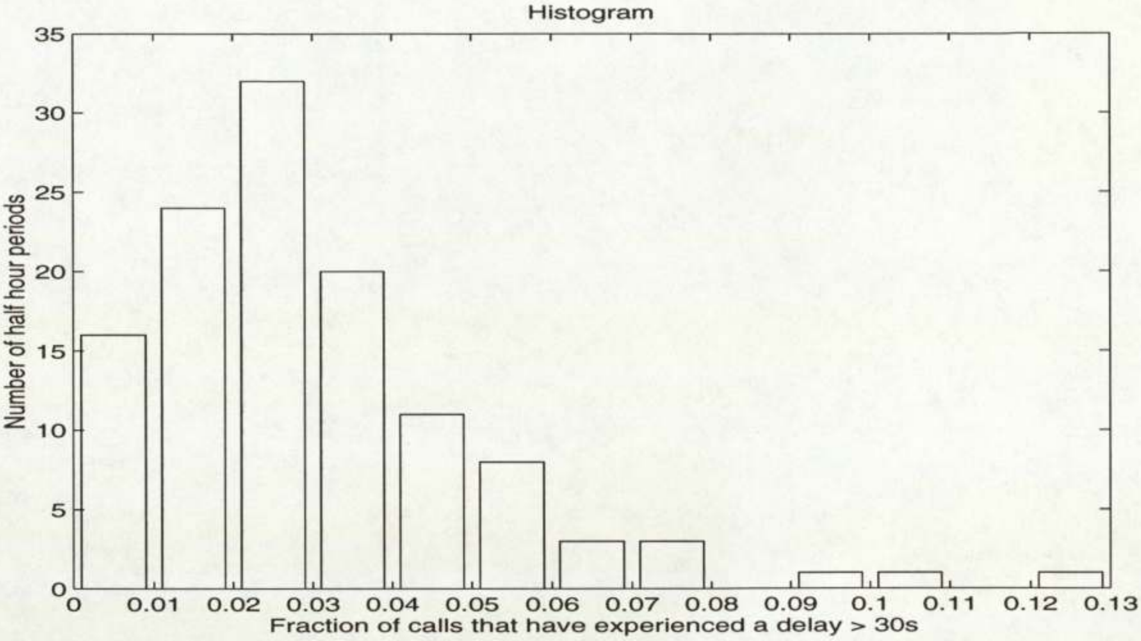


Figure 5.8: Histogram of the number of half hour periods when the service level is not achieved. For every interval (for instance 0.01-0.02, i.e. 1%-2%), we count the numbers of half hour periods for which the fraction of calls, that have experienced a delay greater than 30 seconds, lies in this interval.

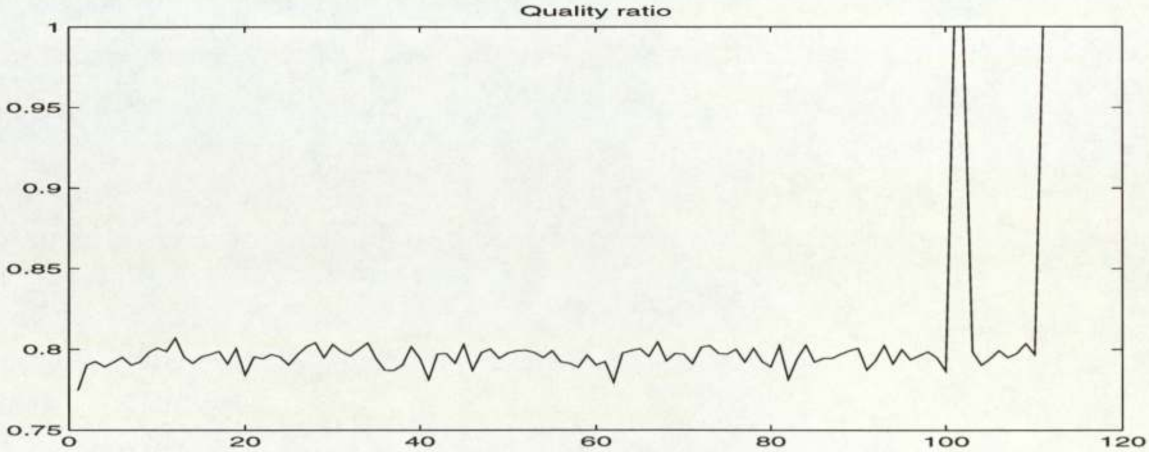


Figure 5.9: Quality of service provided. For each half hour period, the mean competency of the agents answering the incoming calls is computed.

calls. Therefore this pool will first be without any free agents. Moreover if we assume that this pool of agents is the only one to have the skills to answer the calls from a certain queue, then the calls from this queue will have to experience a greater delay as the agents of this pool will be more occupied that the ones of the other pools. Therefore, as our cost function increases the quality ratio, it will allocate more agents to this pool. So we could certainly improve our results by increasing the importance of the measure of quality in the cost function, which will improve the distribution of the agents between the pools and consequently improve the service level.

In Figure 5.10, which shows a typical evolution of the service level throughout a whole week, there is a very interesting characteristic concerning the remaining 10 half hour periods, for which the expected service level is not reached. In all these half hour periods, the activity of the call centre is at his lowest level. The numbers of these half hour periods are around the multiples of 20, which is the number of half hour periods considered in a day. So they correspond either to the start or the end of a day, which are the most quiet periods of the day. This situation is shown in Figure 2.1, which illustrates the activity of each queue of the call centre throughout the whole week. So our model seems to reach its limits when the activity of the call centre is low, which can be partially explained in the case of our simulation. We have used to simulate the arrivals of the calls to the call centre a Poisson process, which is a stochastic process. Hence the number and the pattern of call arrivals fluctuate randomly in each half hour period around the average observed number of calls. Such situation affects much more our model in quiet periods than in busy ones. Indeed, in quiet periods, when the call volume is particularly small, an increase of 2 or 3 calls in an half hour period compared to the expected number of calls could represent an increase of 50%, which is difficult to tackle for the small number of agents predicted in this period.

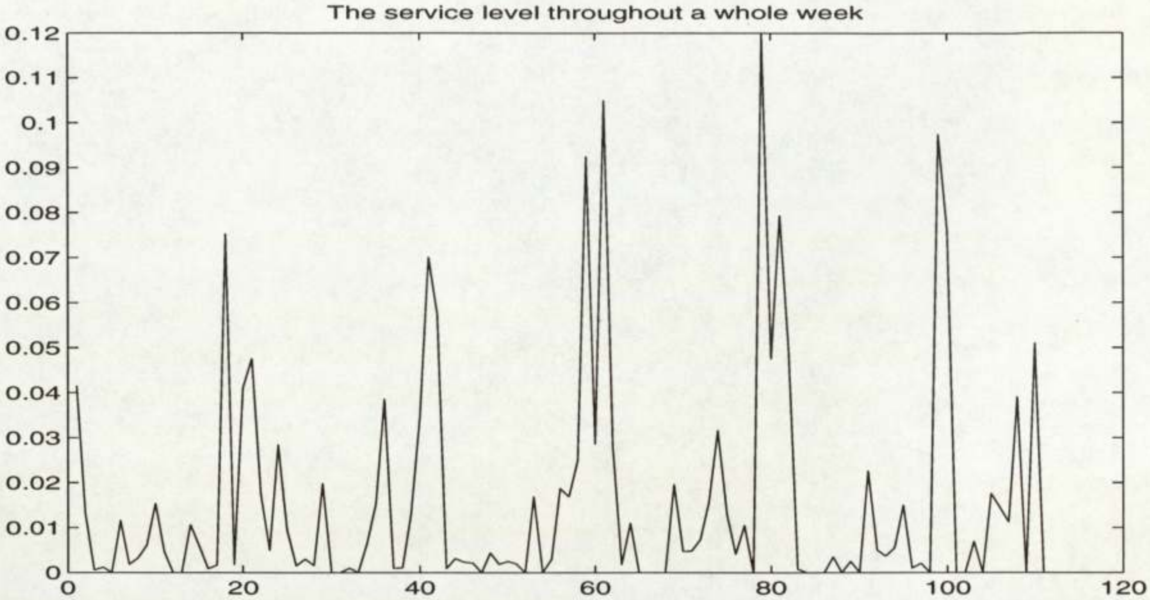


Figure 5.10: The service level computed with the simulation throughout a whole week for the last example. The percentage of calls that have experienced a delay greater than 30 seconds is computed for each half hour period of the week.

5.3.2 Results with another skills matrix

To improve the results, we now make the following assumption: each incoming call can be answered by, at least, two different pools of agents. That means that we modify the skills matrix so that for every queue at least two pools have competencies to answer calls from this queue. This assumption makes the simulation closer to the model we have considered, in the sense there are more agents available to answer each type of call. The skills matrix that we have used is in the table 5.1.

Skills Mix Matrix (unscaled)

Queues	1	2	3	4	5	6
Skill 1	35	100	100	74	0	0
Skill 2	36	0	0	0	100	44
Skill 3	60	**50**	60	0	0	**50**
Skill 4	80	0	51	30	100	0

Table 5.1: The four skills mix factors used in the experiments reported.

In 7 half hour periods, the service level is not reached, as shown in Figure 5.11. This is a strong improvement in the service level. It is also worth noting that only 1.8% of all calls of the whole week are not being answered within 30 seconds. The quality of service provided remains quite high, as the average skill of an agent answering a call is 79%, as shown in Figure 5.12. So we can conclude that our model gives better results when each incoming call can be allocated to, at least, two pools of agents. If this is not the case, we have to improve the distribution of the agents between the pools as it is explained in the following section.

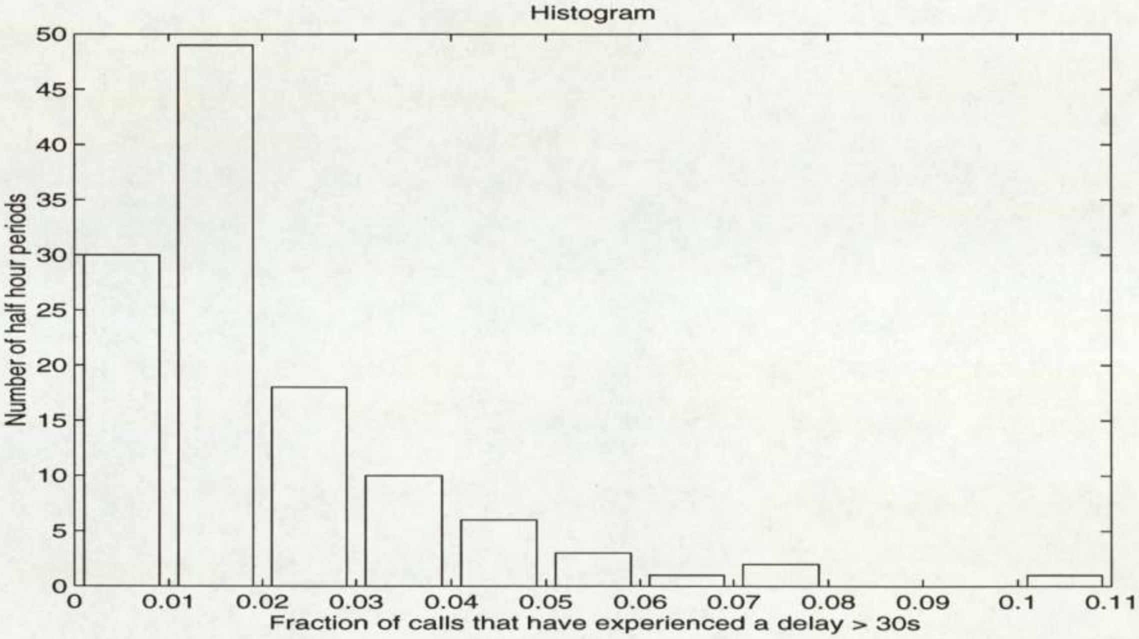


Figure 5.11: Histogram of the number of half hour periods when the service level is not achieved. For every interval (for instance 0.01-0.02, i.e. 1%-2%), we count the numbers of half hour periods for which the fraction of calls, that have experienced a delay greater than 30 seconds, lies in this interval.

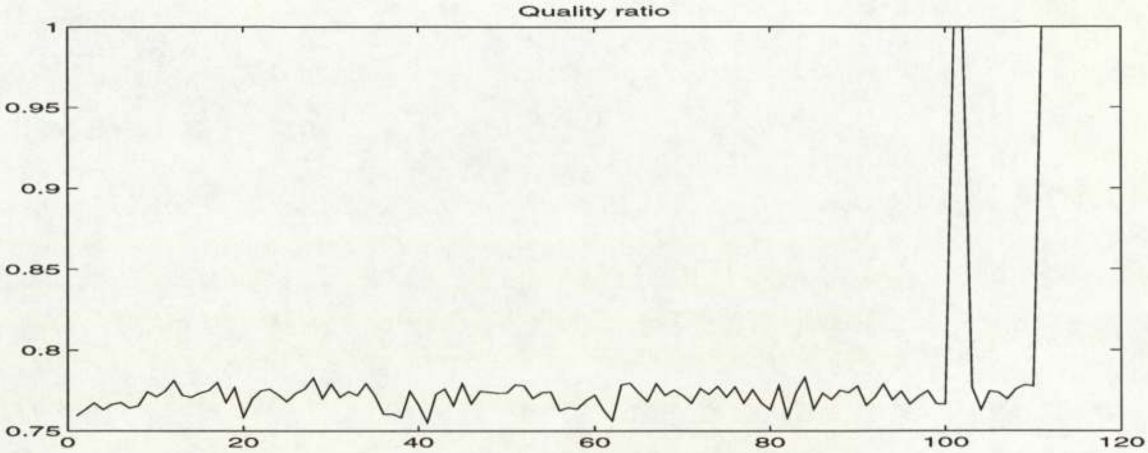


Figure 5.12: Quality of service provided. For each half hour period, the mean competency of the agents answering the incoming calls is computed.

5.3.3 Results with new coefficients in the cost function

Another way to improve the service level is to optimise the distribution of the agents between the pools. This can be done by increasing the weight of the measure of quality in the cost function, thanks to the coefficients reflecting the importance of the different terms of the cost function.

We have changed the coefficients in order to improve slightly the importance of the term representing the quality. This operation doesn't increase the number of agents required in the system. We have used the initial skills matrix, so some types of call can only be answered by only one pool. We note an improvement in the service level: in 11 half hour periods the service level, i.e. 95% of all calls answered within 30 seconds, is not reached, as shown in Figure 5.13, and the global service level for the whole week is 2.25% of all calls experienced a waiting time greater than 30 seconds.The quality of service provided remains quite high, as the average skill of an agent answering a call is 81%, as shown in Figure 5.14.

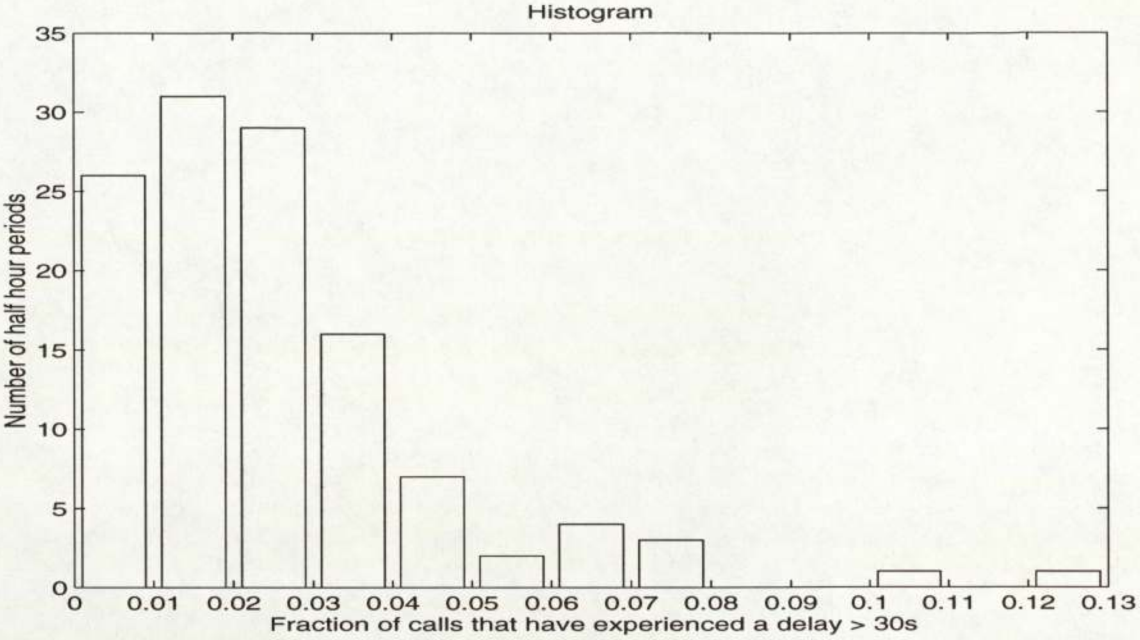


Figure 5.13: Histogram of the number of half hour periods when the service level is not achieved. For every interval (for instance 0.01-0.02, i.e. 1%-2%), we count the numbers of half hour periods for which the fraction of calls, that have experienced a delay greater than 30 seconds, lies in this interval.

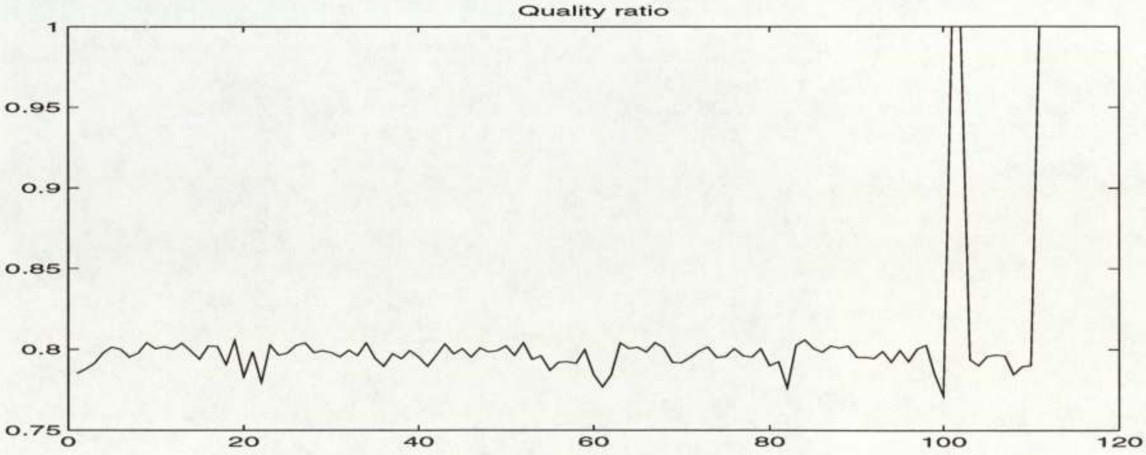


Figure 5.14: Quality of service provided. For each half hour period, the mean competency of the agents answering the incoming calls is computed.

5.4 Conclusions

We have used two methods to validate our approach:

- testing with different data sets

- running some simulations

Of course other methods need to be used to validate more thoroughly our approach and to explore its limits.

The tests with new data sets have shown that the first promising results can also be obtained with other data, that is the gain in the number of staff isn't due to some particular properties of the first data set. By scaling up our data, we have observed the evolution of the results with the activity of the call centre, and that our method still predicts fewer agents for call centres with a very important activity. Even if the percentage of agents gained with our method compared with the traditional one decreases as the activity of the call centre, i.e. the call volume and call duration, increases, the number of agents gained increases with the activity of the call centre. So our method still predicts some interesting economies in the number of staff required.

The simulation has given some interesting results, but not as good as expected since there are some half hour periods, around 15 amongst 120, where the expected service level is not reached. The global service level achieved for the whole week is better than one expected, i.e. 95% percent of calls answered within 30 seconds, as less than 3% of all the calls of the whole week experienced a waiting time greater than 30 seconds. As for the quality of service provided to the calls, the average competence of an agent answering a phone call is 81%, which is quite good. We have nevertheless found some ways of improving the service level in some half hour periods:

- first by building pools of agents so that each type of incoming calls can be answered by two different pools of agents; then in less than 10 half hour periods we fail to achieve the right service level.

- secondly by increasing the importance of the the measure of quality in the cost function predicting the number of agents required in each pool; then in around 10 half hour periods we fail to achieve the right service level.

Obviously the first method is quite constraining and could often be impossible to realise. Nevertheless it is worth noting that one of the strengths of our model is to take into account the multi-skilled aspect of the agents, and consequently by having pools of agents with broad skills profiles we improve the efficiency of our approach.

The second method is much easier to apply. By increasing the importance of the term representing the quality in our cost function, we optimise the distribution of the agents between the pools, which improves the service level. But this method can lead to increase the number of agents required in some pools, which is not what we wanted. However it remains some half hour periods when the service level couldn't easily be reached, these are the half hour periods when the activity of the call centre is low. In such quiet periods, the number of agents required is particularly small and therefore can not easily cope with fluctuations in the call density.

A last remark has to be made. In our simulation, we have assumed that all the customers are infinitely patient, which means that we don't take into account the abandoned calls. Of course, in a real call centre, some of the calls experiencing a delay too important are abandoned, which means that some of the incoming calls won't be answered. This situation leads to improve the service level, since some agents could answer calls waiting for less than 30 seconds instead of calls that would be abandoned.

Chapter 6

People-based approach

6.1 Introduction

In this section we introduce a new aspect of our approach, which is based on individual agents rather than groups of agents. In our first approach, we have decomposed the workforce population into broad skills mix groups. Now, in this second approach, we consider the problem from the perspective of individuals. The theoretical basis of this approach are very close to the ones of the approach used so far and were described in the works carried out by Prof David Lowe and Dr Ian Nabney [1].

Instead of grouping people together into a small number of groups with similar skills profiles, we will now grade each individual member of the workforce with his own unique skills profile. So the incoming calls will be allocated to each individual agent, who is the best qualified agent available. This will lead us to reconsider the meaning of the connectivity matrix, as we now consider connections between an agent and a queue of incoming telephone calls.

The structure of this chapter is very similar to the chapter 2. We will present and justify the construction of a cost function, which differs slightly from the one we have used so far. Then we will detail the results obtained and the conclusions drawn from our works on this particular issue.

6.2 The cost function

Before we construct the various components of the cost function, we need to redefine some mathematical notation, that we have used so far.

Since we now use a stochastic model of the queues, c denotes the total number of agents required in the system to achieve a proper service level and $n_q(t)$ denotes the number of agents needed to answer incoming calls from queue q. c is computed using our stochastic model of the queues and $n_q(t)$ is the fraction of the c agents in the system that are expected to be required to answer calls from queue q.

We also consider that there are P people in the workforce and that person number p is characterised by a skills profile vector s_p. There are a total of Q possible skills that this person could have (corresponding to their ability at answering calls from each of the Q telephone queues). Hence s_p is a Q-dimensional vector. Of course this skills vector is zero everywhere except in the positions which correspond to the types of calls that the person is allowed to answer. We can characterise the whole workforce by forming the complete skills mix matrix which lists the skills profile for each and every person in the workforce. An entry to this matrix, S_{pq} is the ability of person p to be able to answer a telephone call of type (or queue) q. As each person would have his own skills profile, this skills mix matrix is of size $(P \times Q)$.

Finally we have to reconsider the meaning of the connectivity matrix C. The entry c_{pq} denotes a link between queue q and person p. There are two new constraints, following from the perspective of individuals that we are now considering, that have to be imposed on this connectivity matrix.

The first of these constraints is that, now, c_{pq} is the proportion of calls from queue q that person p answers in an half hour period, so it implies $c_{pq} \leq 1$. An efficient way to implement this constraint is to use the penalty function method. If the constraint is violated, we add to the cost function a penalty term:

$$\text{If} \quad c_{pq} > 1, \quad \text{then} \quad E_{new} = E_{old} + \frac{\delta}{2}(c_{pq} - 1)^2$$

The second of these constraints is that, one person can only be answering one call, then the sum of all connectivities to one person must also be either zero or one, i.e. $\sum_{q=1}^{Q} c_{pq} \in \{0, 1\}$. It can also be implemented using the penalty function method:

$$\text{If} \quad \sum_{q=1}^{Q} c_{pq} \neq 0 \quad \text{and} \quad \sum_{q=1}^{Q} c_{pq} \neq 1, \quad \text{then} \quad E_{new} = E_{old} + \frac{\delta}{2}\left(\sum_{q=1}^{Q} c_{pq}\right)^2 \left(\sum_{q=1}^{Q} c_{pq} - 1\right)^2$$

The cost function that we have mentioned is the same that we have used before. It always contains the following terms:

- a term measuring a loss in quality:

$$E_0 = \frac{1}{Q}\sum_{q=1}^{Q}\left(\sum_{p=1}^{P} c_{pq}S_{pq} - n_q\right)^2$$

- a term constraining the sum on each queue to equal the number of calls:

$$E_1 = \frac{1}{Q}\sum_{q=1}^{Q}\left(\sum_{p=1}^{P} c_{pq} - n_q\right)^2$$

- a term corresponding to the minimisation of the number of staff:

$$E_2 = \frac{1}{Q}\sum_{q=1}^{Q}\sum_{p=1}^{P} c_{pq}^2$$

The final cost function we are interested in is obtained by combining all the above terms as:

$$E = E_0 + \alpha E_1 + \beta E_2$$

where α and β are Lagrange multipliers which may be set to specific values which reflect our relative importance of satisfying each of the penalty terms.

And we still need the following constraints to be implemented using the penalty function method:

- no connectivity should be negative:

$$\text{If} \quad c_{pq} < 0, \quad \text{then} \quad E_{new} = E_{old} + \frac{\delta}{2} c_{pq}^2$$

- no call should be answered by people completely unqualified:

$$\text{If} \quad S_{pq} = 0 \quad \text{and} \quad c_{pq} > 0, \quad \text{then} \quad E_{new} = E_{old} + \frac{\delta}{2} c_{pq}^2$$

- the number of agents in the system should be greater than c, that is the minimum number of agents required to achieve the right service level:

$$\text{If} \quad \sum_{q=1}^{Q} \sum_{p=1}^{P} c_{pq} < c, \quad \text{then} \quad E_{new} = E_{old} + \frac{\delta}{2} \left(\sum_{q=1}^{Q} \sum_{p=1}^{P} c_{pq} - c \right)^2$$

An optional constraint can of course be added, if it is necessary, to cap the number of people in some skills mixes. We should then consider the following penalty term:

$$\text{If} \quad \sum_{q=1}^{Q} c_{pq} > A_p, \quad \text{then} \quad E_{new} = E_{old} + \frac{\delta}{2} \left(\sum_{q=1}^{Q} c_{pq} - A_p \right)^2$$

6.3 Results

In this section are presented the first results obtained with the people-based approach. These should be taken as a first attempt to test this new approach, as we hadn't have enough time to explore it thoroughly.

We have run the algorithm through the data of a whole week. As the maximum number of agents predicted by our stochastic model in an half hour period is 83, we have assumed that for each half hour period the number of agents available is 100. Then our algorithm chooses, for each half hour period, amongst all these agents the ones required to achieve a proper service level and the best quality of service. The skills matrix describing the skills profile of each agent is a (100×6) matrix and was designed so that there are agents of very different abilities for each queue. The number of agents required with the people-based approach is the same than the number of people required with the skills mix approach. This situation is depicted on Figure 6.1.

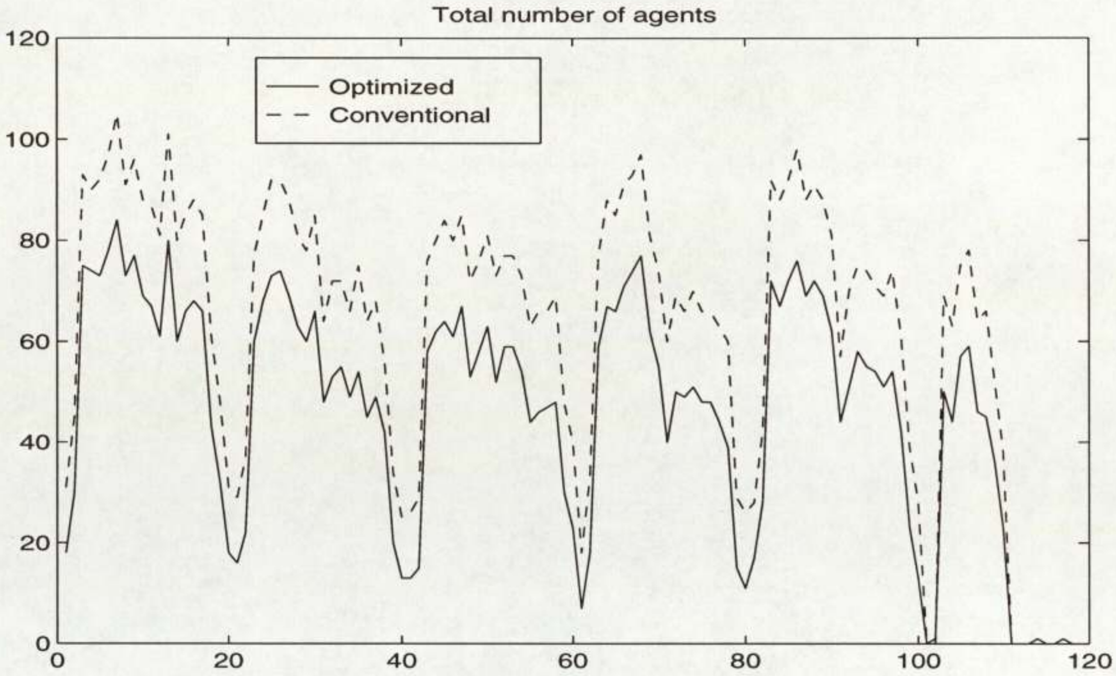


Figure 6.1: Comparison between the number of agents used in the traditional model (the dotted line) and the number of agents used in the people-based approach (solid line).

There are some important difficulties with the people-based approach. These difficulties are a consequence of the large connectivity matrix, that we need to model all the connections between each agent and each queue. First, since there is an important increase in the number of variables, i.e. the connectivities, this may lead to an eventual cost function with many unsuitable minima, which would make our search for a suitable minima more complicated. Secondly a more obvious drawback of the increase in the number of variables is the increase in the time taken by our algorithm to find a suitable minimum to the cost function. With the data set used, which is a relatively small one, it has taken a couple of hours to run the optimisation algorithm. Hence if we use a more important data set, the time taken by the optimisation algorithm will be quite huge. Concerning the search for a suitable minimum, we have encountered some problems with several different values of the coefficients used in the cost function. So this point has to be investigated in more details.

With the results previously obtained, we have run several times the simulation algorithm and computed the average results. Obviously, the results of the simulation are quite disappointing. The right service level, i.e. 95% of all the calls answered within 30 seconds, is not reached in 41 of the 120 half hour periods of the whole week, as shown in Figure 6.2. However, the total percentage of all the calls answered within 30 seconds in the whole week is equal to 4.25%. The quality of service is as good as expected, as the mean competency of the agents concerning the calls they are answering is 81%, as shown in Figure 6.3.

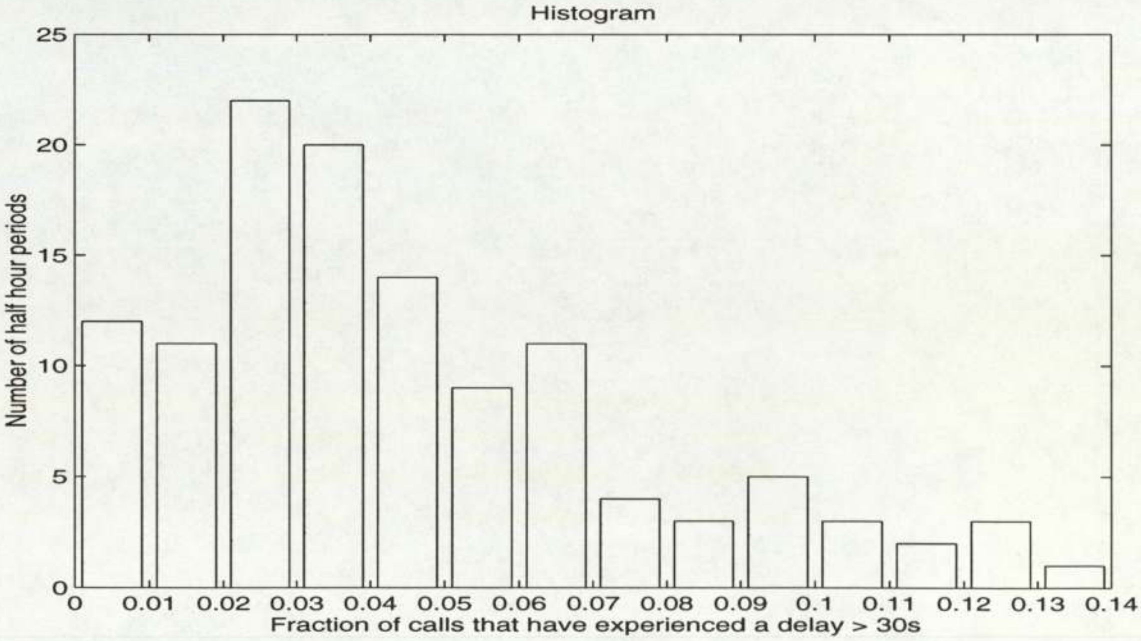


Figure 6.2: Histogram of the number of half hour periods when the service level is not achieved. For every interval (for instance 0.01-0.02, i.e. 1%-2%), we count the numbers of half hour periods for which the fraction of calls, that have experienced a delay greater than 30 seconds, lies in this interval.

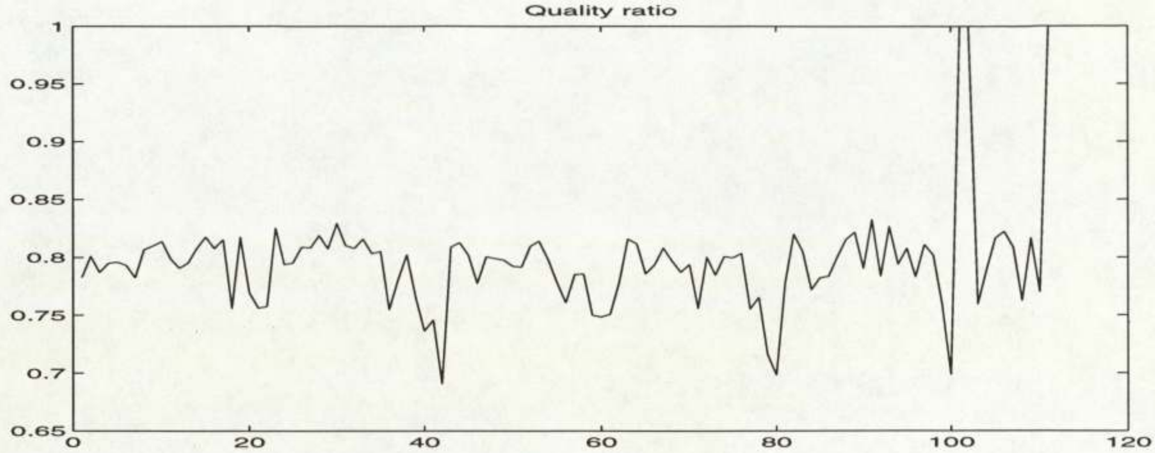


Figure 6.3: The quantification of the quality of service provided by the people-based approach method. For each half hour period, the mean competency of the agents answering the incoming calls is computed.

6.4 Conclusions

The people-based approach appears to be an interesting approach concerning small staffs, as it is more likely in big staffs to have people whose skills profiles are the same and therefore who can be grouped together in skills mix pools. Another point, which makes this approach more adapted to small staffs, is that to consider each agent as an individual implies a huge number of variables, i.e. the connectivities, in our cost function. When the number of variables increases, the time taken to run the algorithm increases and the methods to find the minimum of the cost function become less efficient, as there are many unsuitable minima. We have experienced both problems in our test with a relatively small data set, so this method seems definitively more appropriate to small staffs.

The results obtained are quite disappointing, but this is partially due to the lack of time to explore more deeply this problem. The total number of agents is the same than the one required in the skills mix approach. However, in around 40 half hour periods, the right service level is not reached. Nevertheless the overall service level obtained for the whole week is as good as expected, since only 4.25% of all the calls aren't answered within 30 seconds. The quality of answer provided to the calls is quite good, as the average competency of an agent answering a call is 81%. Obviously more work has to be done on this issue, especially with small data sets which may require only around 30-40 agents. Then it is really possible to explore all the properties of this approach.

Chapter 7

Conclusions

7.1 Current state of the project

This project has started on the basis of the feasibility study made by Prof David Lowe and Dr Ian Nabney [1]. Its purpose was to investigate some assumptions made and to go further in the validation of the method introduced in this first study.

The conclusions drawn from the feasibility study are presented in the section 2.4. They are quite encouraging as, compared to the traditional method the proposed approach appears to predict fewer staff required, whilst still returning a good level of overall performance in term of the percentage of calls answered. The method devised also provides a quantitative index of the quality of service provided to the incoming calls. However these results were obtained using some assumptions that needed to be investigated.

The first assumption that we have considered in this project is that there is an unlimited pool of people in each skills mix. Obviously this is not the case in practice for all the more highly qualified profiles. Therefore some methods were devised to cap the number of agents in some skills mixes. We have found that the penalty function method gives the best results and is not too computationally expensive. The effects of capping on the number of staff, the service level and the quality of service are investigated through some examples. It appears that it is possible to maintain a good service level while capping the number of people in some skills mixes. In this situation, the number of agents required increases in the skills mixes whose number of staff is uncapped so that a high proportion of calls can be answered. An effect of capping the number of agents might be a decrease of the quality of service. This is explained by the fact that the number of staff is often limited in the most qualified skills mixes, so that the less qualified agents are more numerous and therefore more likely to answer phone calls.

The second assumption considered is that the calls are uniformly distributed across the half hour period and that all calls take the same length of time to be answered. These assumptions are quite inaccurate, since they don't take into account the fluctuations in the arrival pattern of the calls and in the service time distribution. To explore and develop this assumption, we have used queueing theory which gives us the mathematical tools that we need to build our own stochastic model of the queues. We have made same assumptions about arrival pattern and service time distribution as the Erlang's C model, which is what is currently used. To model our system, we have chosen a Poisson process as the arrival pattern and a hyper-exponentially distributed service time. Once this model is chosen, the number of people required in the system to achieve a proper service level can be computed. We then run our optimisation algorithm which shares this total number of agents between the different pools in order to maximise the quality of service. The number of people required in our system increases, since the stochastic modelling of the queues results in assigning more than the expected number of agents to cope with fluctuations in the call density. However this increase in the number of staff remains quite small and our model still requires much fewer staff than the traditional one.

Some other issues are also investigated in this project. The first one concerns tests made with new data sets in order to observe how the results vary with different data sets. To analyse the evolution of the results through the size of the data, we have scaled up our initial data set. The conclusion is that the percentage of agents gained with our method compared with the traditional one decreases as the call centre activity increases, but the number of agents gained increases. Thus our method still allows useful gains in the number of staff with an important activity of the call centre, but not as important as expected. Another aspect of this issue is to try our algorithm on new data sets to check if the results of our method aren't due to the particular nature of our usual data set. After running our algorithm through two new data sets, we have seen that the results are also quite good for other data. So this gives encouraging results on the reliability of our method with different data sets.

Another issue investigated is the validation of our method through simulation. We have run an algorithm, that simulates the arrival pattern and the service time distribution in a call centre. To measure the performances of our approach, we have computed the service level for each half hour period and for the whole week and also the quality of service provided to the customers. The service level will be reached if 95% of all calls are answered within 30 seconds. The service level for the whole week is reached and even better than expected. The average quality of service in each half hour period is also slightly better than expected. These two results are quite encouraging and are an important steps towards a final implementation of our approach in a real case. However in some half hour periods, about ten, the service level is not reached. This happens when the activity of the call centre is small and in such situations only 3 calls not answered within 30 seconds represent a relatively high percentage of all the calls, around 10%. So these are isolated problems which don't really call into question our approach.

The last issue discussed is the people-based approach, which has been already introduced in the feasibility study on which this project is based. We have detailed the point of view adopted in this approach and justified it. It appears that this approach requires the same number of agents as the skills mix approach. However the results of the simulation concerning the service level aren't as good as these obtained in the skills mix approach. Some difficulties are linked to this approach, they are due to the increase in the number of variables, which represent a connection between a agent and a queue. It is important noting that the people-based approach has not been thoroughly investigated in this project and that, consequently, more work has to be done on this issue.

7.2 Further points to investigate

Some work has still to be done on some of the issues of this new approach. We shall give in this section the main ones:

1. other arrival patterns than the Poisson process can be investigated. For example in the case of people phoning in call centre in response to a television advertisement, all the calls arrive mainly during the same small period of time following the advertisement.

2. some other distributions, different from the exponential one, can be used to model the service time. For instance a normal distribution with the mean and standard deviation computed using some data about the call duration of calls could be more appropriate.

3. some work had to be done on the simulation in order to investigate how we can improve our results and in order to make it closer to the real behaviour of a call centre. For example, the allocation of incoming calls to agents can be optimised to achieve the best service level and to improve the quality of service provided to the customers.

4. abandoned calls can be introduced in our model, this could decrease the number of staff required.

5. the meaning of the quality ratio can be investigated as it is an important measure of the efficiency of our method. Possible models of the effects of skill on the call answering can be explored. For instance, a less skilled agent may come up with the same answer in a longer period of time.

6. the people-based approach has to be investigated in more detail as it is an important aspect of our general approach to the resource allocation problem.

Appendix A

Scaled Conjugate Gradients Algorithm

In this appendix we shall describe in some detail the Scaled Conjugate Gradients Algorithm. We consider the minimisation of the non-linear function f, whose variable is the vector x. This algorithm is based on a search through the variables space consisting of a succession of steps of the form:

$$x_{j+1} = x_j + \triangle x_j$$

where x_j is the variable vector at the step j, j labels the iteration steps and $\triangle x_j$ represents the increment of our variable. Each steps can be divided in two parts:

- first we must decide the direction in which to move
- secondly we must decide how far to move in that direction

Suppose that at step j, the current variable is x_j and we wish to consider a particular search direction d_j through the variables space. The minimum along the search direction then gives the next value for the variable:

$$x_{j+1} = x_j + \alpha_j d_j$$

where the parameter α_j is chosen to minimise

$$f(\alpha) = f(x_j + \alpha d_j)$$

This gives us an automatic procedure for setting the step length, once we have chosen the search direction.

This is a summary of the key steps of the algorithm:

1. Choose an initial variable vector x_1

2. Evaluate the gradient vector $g_1 = \nabla f$ at the point x_1, where ∇ denotes the gradient operator in the variables space
   Set the initial search direction $d_1 = -g_1$
   Set the initial scale parameter $\lambda_1 = 1$ and $f_1 = f(x_1)$

3. At step j:
   Evaluate $\delta_j = d_j^T H d_j + \lambda_j \|d_j\|^2$, H is the Hessian matrix of the function f
   If $\delta_j < 0$ then $\lambda_j = 2\left(\lambda_j - \frac{\delta_j}{\|d_j\|^2}\right)$
   Compute the step length $\alpha_j = \frac{-d_j^T g_j}{\delta_j}$
   Derive the new value of the variable $x_{j+1} = x_j + \alpha_j d_j$
   Update the value of the function $f_{j+1} = f(x_{j+1})$
   Introduce a comparison ratio $\triangle_j = \frac{2(f_{j+1} - f_j)}{\alpha_j d_j^T g_j}$, which gives a measure of the approximation made in this algorithm
   If $\triangle_j < 0$ then $x_{j+1} = x_j$, i.e. don't take the step

4. Test to see if the stopping criterion is satisfied

5. Evaluate the new gradient vector $g_{j+1} = \nabla f$ at the point x_{j+1}

6. Adjust λ_{j+1} according to comparison ratio:
   if $\triangle_j > 0.75$ then $\lambda_{j+1} = \frac{\lambda_j}{2}$
   if $\triangle_j < 0.25$ then $\lambda_{j+1} = 4\lambda_j$
   otherwise $\lambda_{j+1} = \lambda_j$

7. Evaluate the new search direction:
   $d_{j+1} = -g_{j+1} + \beta_j d_j$,

with $\beta_j = \frac{g_{j+1}^T(g_{j+1}-g_j)}{d_j^T(g_{j+1}-g_j)}$ Hestenes-Stietel formula

We have used in the step 3 H, the Hessian matrix of the function f, but only on the form of the Hessian multiplied by a vector d_j. So the quantity of interest is not the Hessian matrix H itself, but the product of H with the vector d_j. Instead of computing the Hessian as an intermediate step, we have used an efficient approach to evaluating $d_j^T H$ directly, which requires significantly less operations. We first note that:

$$d_j^T H \equiv d_j^T \nabla(\nabla f))$$

We can then estimate the right-hand side of this equation using finite differences to give:

$$d_j^T \nabla(\nabla f)) = \frac{\nabla f(x_j + \epsilon d_j) - \nabla f(x_j)}{\epsilon}$$

where ϵ denotes a small coefficient, such that we can apply the finite differences with the vector ϵd_j.

Appendix B

Erlang's C model

B.1 Introduction

B.1.1 The Poisson distribution

The Poisson distribution with the parameter λ is defined by the formula:

$$P_x(\lambda) = \frac{e^{-\lambda} \times \lambda^x}{x!} \qquad x = 0, 1, 2...$$

It represents the probability that a random variable X takes on the value x:

$$Prob(X = x) = P_x(\lambda)$$

B.1.2 The Poisson process

It's a model for describing the occurrence of random events in time. There are three postulates that specify a Poisson process with rate $\lambda > 0$:

1. The probability that an event occurs during any small time interval $[t, t + \delta t]$ is equal to $\lambda \delta t + o(\delta t)$.

2. The probability that more than one event occurs during any small time interval $[t, t + \delta t]$ is equal to $o(\delta t)$.

3. The occurrence of events after any time t is independent of the occurrence of events up to time t.

The random variable X representing the number of events occurring by time t in a Poisson process has a Poisson distribution, with the parameter λt:

$$P_x(t) = \frac{e^{-\lambda t} \times (\lambda t)^x}{x!} \qquad\qquad x = 0, 1, 2, ...$$

B.1.3 Birth-and-death process

We shall consider a 'population' which is simultaneously gaining new members through birth and losing old members through death. We also assume that the birth rate, λ_n, and the death rate, μ_n, depend upon n, the size of the population. We denote each state of the population, whose size is n, by E_n.

A system, which has a population of n elements at time t, is said to be described by a birth-and-death process, if there exist non-negative birth rates, $\{\lambda_n, n = 0, 1, 2, ...\}$ and non-negative death rates, $\{\mu_n, n = 1, 2, ...\}$, such that the following postulates are satisfied:

- State changes are only allowed from state E_n to state E_{n+1}, $n \geq 0$, or from state E_n to state E_{n-1}, $n \geq 1$.
- The probability that the transition $E_n \rightarrow E_{n+1}$ (birth) occurs during any small time interval $[t, t + \delta t]$ is equal to $\lambda_n \delta t + o(\delta t)$.
- The probability that the transition $E_n \rightarrow E_{n-1}$ (death) occurs during any small time interval $[t, t + \delta t]$ is equal to $\mu_n \delta t + o(\delta t)$.
- The probability that more than one transition, either birth or death, occurs during any small time interval $[t, t + \delta t]$ is equal to $o(\delta t)$.

In general, it's difficult to find the time-dependent solutions of a birth-and-death process. However, we shall consider a particular case: the steady state. This state is characterised by the following statement:

$$\forall\, n \ \ \lim_{t \to +\infty} P_n(t) = p_n$$

Then we derive the following results:

- $\forall\, n \geq 1 \ \ p_n = (\prod_{k=1}^{n} \frac{\lambda_{k-1}}{\mu_k}) p_0$

- Since $\sum_{n=0}^{+\infty} p_n = 1, \ \ S = 1 + \sum_{n=1}^{+\infty} (\prod_{k=1}^{n} \frac{\lambda_{k-1}}{\mu_k}) = p_0^{-1}$.

B.1.4 Exponential random variables

A continuous random variable X has an exponential distribution with parameter $\lambda > 0$, if its density function f is defined by:

- $f(x) = \lambda e^{-\lambda x} \qquad x > 0$

- $f(x) = 0 \qquad x \leq 0$

The distribution function F is then given by:

- $F(x) = 1 - e^{-\lambda x} \qquad x > 0$

- $F(x) = 0 \qquad x \leq 0$

Some properties of the exponential distribution:

- $E[X] = \frac{1}{\lambda}$

- $Var[X] = \frac{1}{\lambda^2}$

- $E[X^k] = \frac{k!}{\lambda^k} \qquad k = 0, 1, 2, ...$

- $P[X > t + h | X > t] = P[X > h] \qquad t > 0 \quad h > 0$

- If the inter-arrival times of customers to a queueing system are independent, identically distributed, exponential random variables, each with mean $1/\lambda$, then the number of arrivals, Y_t, in any time interval of length $t > 0$, has a Poisson distribution with parameter λt, that is,

$$P[Y_t = k] = \frac{e^{-\lambda t} \times (\lambda t)^k}{k!} \qquad k = 0, 1, 2, ...$$

B.2 Erlang C model

B.2.1 Assumptions and definitions

This model is based upon the following assumptions [10]:

- random (Poisson) arrival pattern and λ is the corresponding arrival rate
- exponentially distributed service time and μ is the corresponding service rate
- c servers
- infinitely many waiting slots in the system
- the longest-waiting call is always the one selected when a server becomes available
- all customers are infinitely patient

It is useful to introduce this two parameters:

- $a = \frac{\lambda}{\mu}$: offered load in erlangs
- $T = \frac{1}{\mu}$: average handle time

In addition, we establish the following notations:

- t: maximum acceptable delay
- $F(t)$: fraction of calls that experience a delay greater than t
- W: average delay for all calls, including those whose delay is 0
- $B = B(c, a) = \frac{\frac{a^c}{c!}}{\sum_{n=0}^{c} \frac{a^n}{n!}}$, Erlang's B function
- $C = C(c, a) = \frac{c \times B(c,a)}{c - a \times (1 - B(c,a))}$, Erlang's C function

With such assumptions, the birth rates are equal to: $\lambda_n = \lambda$

And the death rates are equal to:

- $\mu_n = n\mu \qquad n \leq c$
- $\mu_n = c\mu \qquad n \geq c$

The probabilities can then be computed:

- $p_0 = [\sum_{n=0}^{c-1} \frac{a^n}{n!} + \frac{a^c}{c!} \times \frac{c}{c-a}]^{-1}$
- $p_n = \frac{a^n}{n!} p_0 \qquad n \leq c$
- $p_n = \frac{a^n}{c! c^{n-c}} p_0 \qquad n \geq c$

B.2.2 Formulas for service time and average delay

The service level is computed using:

$$F(t) = Ce^{-(c-a)t/T}$$

where $F(t)$ is the fraction of calls whose delay is longer than the maximum acceptable delay t, C is the Erlang's C function, n is the number of agents, T is the average handling time and a is the offered load in erlangs.

The average delay for all calls is given by:

$$W = \frac{C \times T}{c - a}$$

where C is the Erlang's C function, n is the number of agents, T is the average handling time and a is the offered load in erlangs.

Appendix C

Stochastic modelling of the queues

In this appendix, some points introduced in chapter 4 are discussed in more detail.

C.1 Calculation of the percentile value

In chapter 4, it was said that since the distribution of the queueing time, d, approaches that of an exponential distribution, one can use the following formula to calculate the r^{th} percentile value of d:

$$\pi_d(r) = E[d] \times \ln\left(\frac{100}{100-r}\right) = W_d \times \ln\left(\frac{100}{100-r}\right)$$

In order to justify this result, we shall consider X, an exponentially distributed random variable, and $\pi(r)$, its r^{th} percentile value, defined by:

$$P(X \leq \pi(r)) = \frac{r}{100}$$

Since X is an exponentially distributed random variable, whose parameter is denoted λ, we can write:

$$P(X \leq \pi(r)) = 1 - e^{-\lambda\pi(r)} = 1 - e^{-\frac{\pi(r)}{E[X]}}$$

We have seen in appendix B that the expected value of an exponentially distributed random variable with parameter λ is equal to $\frac{1}{\lambda}$. So:

$$1 - e^{-\frac{\pi(r)}{E[X]}} = \frac{r}{100} \quad \Longleftrightarrow \quad e^{-\frac{\pi(r)}{E[X]}} = \frac{100-r}{100}$$

This implies the following result:

$$\pi(r) = E[X] \times \ln\left(\frac{100}{100-r}\right)$$

C.2 Erlang's C function

In chapter 4, we have introduced Martin's estimate, which uses Erlang's C function $C(c,a)$. This function is defined for a queueing system, where c is the number of servers and $a = \frac{\lambda}{\mu}$, where λ is the average arrival rate of calls to the system and μ is the average service rate per server, is the offered load in erlang. The following formulas defined Erlang's C function:

$$C = C(c,a) = \frac{c \times B(c,a)}{c - a \times (1 - B(c,a))} \qquad \text{Erlang's } C \text{ function}$$

$$\text{where} \quad B = B(c,a) = \frac{\frac{a^c}{c!}}{\sum_{n=0}^{c} \frac{a^n}{n!}} \qquad \text{Erlang's } B \text{ function}$$

This last formula defines the Erlang's B function $B(c,a)$, but is not useful for computation. One excellent method for computing $B(c,a)$ is the following iterative formule:

- $B(0,a) = 1$
- $B(k,a) = \frac{a \times B(k-1,a)}{k + a \times B(k-1,a)} \qquad k = 1,2,3,...,c$

C.3 Expected value of an hyper-exponentially distributed random variable

We shall consider X, an hyper-exponentially distributed random variable with parameters $\mu_n, n = 1,2,3,...,N$. So the density function of X is given by:

$$f_x(t) = \sum_{n=1}^{N} \alpha_n \mu_n e^{-\mu_n t} \qquad \text{with} \quad \sum_{n=1}^{N} \alpha_n = 1$$

The expected value of X is defined as follow:

$$E[X] = \int_{-\infty}^{\infty} f_x(t)dt = \int_{-\infty}^{\infty} \sum_{n=1}^{N} \alpha_n \mu_n e^{-\mu_n t} dt = \sum_{n=1}^{N} \alpha_n \int_{-\infty}^{\infty} \mu_n e^{-\mu_n t} dt = \sum_{n=1}^{N} \frac{\alpha_n}{\mu_n}$$

Since the expected value of an exponentially distributed variable with parameter μ_n is $\frac{1}{\mu_n}$:

$$\int_{-\infty}^{\infty} \mu_n e^{-\mu_n t} dt = \frac{1}{\mu_n}$$

So the expected value of s, the service time, is equal to:

$$E[s] = \sum_{q=1}^{Q} \frac{\alpha_q}{\mu_q}$$

Appendix D

Outline algorithm

This appendix displays the full algorithm which was used to produce the figures obtained in this thesis. The code is written in the interpreted matrix manipulation language MATLAB.

D.1 Top-level optimisation program

The following code is the top-level program used to read in information, set up the costs and minimise the cost function iteratively until a minimum has been found. It includes a method of capping the number of people in certain pools and the stochastic modelling of the queues.

```
% Optimise staff scheduling

% Ian Nabney 13 August 1996 and Gilles Le Corroller 23 August 1997

% Read in the skills matrix
load skills;
% Rescale to range 0-1.
skills = skills ./100.0;
% Only use the first four skills profiles to make it interesting
% In the people-based approach this line should be removed
skills = skills(1:4, :);

% Read in the data matrix describing the activity of the queues
load volume;
% time_steps_a_day is the number of half hour periods in a day
time_steps_a_day = 20;

% Compute what should be the number of agents answering calls
% from each queue, according to the stochastic modelling of the queues
H_q = zeros(120,8);
H_q = H_ratio(volume,time_steps_a_day);
% queue contains the required number of agents in each queue
queue = H_q(:,1:6);
% server contains the total number of agents required in the system
server = H_q(:,7);
```

```
% Read in the agents matrix, that contains the capping numbers
% for each queue in each half hour period
load H_ag.dat;

rand('seed', 42);

options = foptions;
%options(1) = 1; % Display error values
%options(9) = 1;  % Check gradient calculations
options(14) = 10; % Number of training cycles

% total number of half hour periods for the whole week
num_time_steps = size(queue, 1);
% number of skills mixes
num_sk = size(skills, 1);
% number of queues
num_q = size(skills, 2);

% Initialise results matrices
staff = zeros(num_time_steps, num_sk);
calls = zeros(num_time_steps, num_q);
quality = zeros(num_time_steps, num_q);

tic;
for i = 1:1:num_time_steps
  fprintf('Time step %d\n', i);
  % Initialise connection matrix
  con = rand(size(skills));

  agents_i = H_ag(i,:);
  queue_i = queue(i, :);
  server_i = server(i);

  % Set hyperparameters on Lagrange multipliers for constraints
  alpha = 1.0; % If people-based approach then 1 else 0
  beta  = 10.0; % Total sum on queue equals number of servers
  gamma = 1.0; % Minimize number of staff
  delta = 1.0; % Total number of servers in the system
  penalty = 1;  % Coefficient of the penalty function method

  for u = 1:1:10
    penalty  = penalty * 4;
    % Pass vector form of connection matrix to optimizer
    con = scg('costs', con(:)', options, 'dcosts', queue_i, ...
          skills, agents_i, server_i, alpha, beta, gamma, delta, penalty);
    con = reshape(con, num_sk, num_q);
  end

  % Make sure all values are positive
  con = max(con, zeros(size(con)));

  % This line is for the people-based approach
  % It rounds the connectivity to 0 or 1
  % con = round(con);

  % Should round up the number of agents answering queues to integer values
  % sum function does column sum
  calls(i, :) = ceil(sum(con));
  for j = 1:num_q
    %    fprintf('Number of calls in queue %d is %f\n', ...
    % round(j), calls(i, j));
  end

  % Should round up the number of agents in each pool to integer values
  staff(i,:) = ceil(floor(sum(con')*100)/100);
  for j = 1:num_sk
  %    fprintf('Number of staff in skills_mix %d is %f\n', ...
  % round(j), staff(i, j));
  end
  fprintf('Total staff is %f\n', sum(staff(i, :)));

  % Calculate quality on a queue by queue basis
  quality(i, :) = sum(skills.*con);

end
toc;

save 'staff.dat' staff -ascii;
save 'calls.dat' calls -ascii;
save 'quality.dat' quality -ascii;
```

D.2 Code for the cost function

The following code evaluates the costs asociated with the connectivity.

```
function q = costs(con, calls, skills, agents, server, alpha, beta, gamma,..
            delta, penalty)

% COSTS Calculates cost of connectivity matrix
%
% This has been coded for maximal clarity rather than speed, so
% uses few vectorized operations.
% However sometimes the vectorised version of operations is also mentionned

% Find number of skills
num_sk = size(skills, 1);
% Find number of queues
num_q = size(skills, 2);

% Turn (local) vector into more user-friendly matrix
con = reshape(con, num_sk, num_q);

% Calculate quality cost
d1 = 0;
for j = 1:num_q
  d2 = 0.0;
  for i = 1:num_sk
    d2 = d2 + skills(i, j)*con(i, j);
  end
  d1 = d1 + (d2 - calls(j)).^2;
end
q = d1/num_q;

% Add cost to match the number of servers in each queue
d1 = 0.0;
for j = 1:num_q
  d2 = 0.0;
  for i = 1:num_sk
    d2 = d2 + con(i, j);
  end
  d1 = d1 + (d2 - calls(j)).^2;
end
q = q + (d1 * beta)/num_q;

% Minimise number of staff
d1 = 0.0;
for i = 1:num_sk
  for j = 1:num_q
    d1 = d1 + con(i,j).^2;
  end
end
q = q + ((gamma * d1) / num_q);

% Add cost to match the total number of servers
d1 = 0.0;
for j = 1:num_q
  for i = 1:num_sk
    d1 = d1 + con(i,j);
  end
end
d1 = (d1 - server)^2;
q = q + (d1 * delta)/num_q;

% Penalty function method
% Keep connectivities positive
d1 = 0.0;
for i = 1:num_sk
  for j = 1:num_q
   if ( con(i,j) < 0 ) d1 = d1 + penalty/2*con(i,j)^2; end
  end
end
% Optimised version of the penalty function method
% d1 = sum(sum((con<0)*penalty/2.*con.^2));
q = q + d1;

% Penalty function method
% If Spq = 0, then cpq should be equal to 0
d1 = 0.0;
for i = 1:num_sk
  for j = 1:num_q
   if (skills(i,j) == 0) & (con(i,j) > 0) d1 = d1 + penalty/2*con(i,j)^2; end
  end
end
% Optimised version of the penalty function method
% d1 = sum(sum((skills == 0).*(con>0)*penalty/2.*con.^2));
q = q + d1;

% Penalty function method
% The total number of agents should be greater than c
d1 = 0.0;
for i = 1:num_sk
  for j = 1:num_q
    d1 = d1 + con(i,j);
  end
end
if d1 < server d1 = penalty/2*(d1-server)^2;
% Optimised version of the penalty function method
% d1 =  ((sum(sum(con))-server) < 0)*penalty/2*(sum(sum(con)) - server)^2;
q = q + d1;

% Penalty function method
% Connection <= 1.  Put alpha = 0 to ignore this (if
% connections coded on a skills rather than person basis)
% Put alpha = 1 if it is the people-based approach
d1= 0.0;
for i = 1:num_sk
  for j = 1:num_q
    if con(i,j) > 1
       d1 = d1 + penalty/2*(con(i,j)-1)^2; end
  end
end
% Optimised version of the penalty function method
% d1 = sum(sum((con>1).*penalty/2.*(con-1).^2));
q = q + alpha*d1;
```

```
% Penalty function method
% Sum of connections to one person = 0 or 1
% Put alpha = 0 if it is the skills mix approach
% Put alpha = 1 if it is the people-based approach
d1= 0.0;
for i = 1:num_sk
    if ( sum (con(i,:)') ~= 0 ) & ( sum( con(i,:)')  ~= 1 )
        d1 = d1 + penalty/2*sum(con(i,:)')^2*(sum(con(i,:)')-1)^2; end
end
% Optimised version of the penalty function method
% d1 = sum(((sum(con')~=0)&(sum(con')~=1))*...
%      penalty/2.*sum(con').^2.*(sum(con')-1).^2);
q = q + alpha*d1;

% Penalty function method
% Add a constraint term
% Cap the number of people in certain skills mix area
d1 = 0.0;
for i = 1:num_sk
  if agents(i) > 0
    if sum (con(i,:)') > agents(i)
       d1 = d1 + penalty/2*(sum(con(i,:)')-agents(i))^2; end
  end
end
% Optimised version of the penalty function method
% d1 = sum((agents > 0).*((sum(con')-agents) > 0)*...
% penalty/2.*(sum(con')-agents).^2);
q = q + d1;
```

D.3 Code for the gradient of the cost function

The following code evaluates the gradient of the cost function.

```
function dq = dcosts(con, calls, skills, agents, server, alpha, beta, ...
                     gamma,delta, penalty)

%DCOSTS Calculates partial derivatives of cost of connectivity matrix
%
% This has been coded for maximal clarity rather than speed, so
% uses few vectorized operations.
% However sometimes the vectorised version of operations is also mentionned

% Find number of skills
num_sk = size(skills, 1);
% Find number of queues
num_q = size(skills, 2);

% Turn (local) vector into more user-friendly matrix
con = reshape(con, num_sk, num_q);

dq = zeros(size(con));

for j = 1:num_q
  d2 = 0.0;
  for i = 1:num_sk
    d2 = d2 + skills(i, j)*con(i, j);
  end
  d2 = 2*(d2 - calls(j))/num_q;

  d3 = 0.0;
  for i = 1:num_sk
   d3 = d3 + con(i, j);
  end
  d3 = 2*(d3 - calls(j))/num_q;

  d4 = 0.0;
  for i = 1:num_sk
    for k = 1:num_q
        d4 = d4  + con(i,k);
    end
  end
  d4 = 2*(d4 - server)/num_q;

  for i = 1:num_sk
    dq(i, j) = d2 * skills(i, j);    % Quality
    dq(i, j) = dq(i, j) + beta*d3;   % Match number of calls in this queue
    dq(i, j) = dq(i, j) + gamma*2*con(i, j)/num_q; % Minimize staff
    dq(i, j) = dq(i, j) + delta*d4;  % Match number of servers in the system
  end
end

% Penalty function method
% Connectivities positive
for i = 1:num_sk
  for j = 1:num_q
    if  con(i,j) < 0  dq(i,j) = dq(i,j) + penalty*con(i,j); end
  end
end
% Optimised version of the penalty function method
% dq = dq + (con<0)*penalty.*con;

% Penalty function method
% If Spq = 0, then cpq should be equal to 0
for i = 1:num_sk
  for j = 1:num_q
    if (skills(i,j) == 0) & (con(i,j) > 0)
       dq(i,j) = dq(i,j) + penalty*con(i,j);
    end
  end
end
% Optimised version of the penalty function method
%dq = dq + (skills == 0).*(con>0)*penalty.*con;

% Penalty function method
% The total number of agents should be greater than c
d4 = 0.0;
if sum(sum(con)) < server d4 =  penalty*(sum(sum(con)) - server);
                           for i = num_sk
                             for j = num_q
          dq(i,j) = dq(i,j) + d4;
      end
    end
end
% Optimised version of the penalty function method
% dq = dq + penalty*ones(num_sk,6)*((sum(sum(con))-server) < 0)*...
% (sum(sum(con))-server);

% Penalty function method
% Connectivities <= 1
% Put alpha = 0 if it is the skills mix approach
% Put alpha = 1 if it is the people-based approach
for i = 1:num_sk
  for j = 1:num_q
    if  con(i,j) > 1  dq(i,j) = dq(i,j) + alpha*penalty*(con(i,j)-1); end
  end
end
% Optimised version of the penalty function method
%dq = dq + alpha*(con>1)).*penalty.*(con-1);

% Penalty function method
% Sum of connections for one person = 0 or 1
% Put alpha = 0 if it is the skills mix approach
% Put alpha = 1 if it is the people-based approach
for i = 1:num_sk
  for j = 1:num_q
    if  (sum(con(i,:)') ~=0) & (sum(con(i,:)') ~=1)  dq(i,j) = dq(i,j) +...
                     alpha*penalty*(sum(con(i,:)')*(sum(con(i,:)')-1)^2+...
                                  sum(con(i,:)')^2*sum((con(i,:)')-1)); end
  end
end
% Optimised version of the penalty function method
% dq = dq + alpha*(ones(6,1)*(((sum(con')~=0)&(sum(con')~=1))*...
%      penalty.*(sum(con').*(sum(con')-1).^2+sum(con').^2.*(sum(con')-1))))'

% Penalty function method
% Cap the number of people in certain skills mixes
for i = 1:num_sk
  for j = 1:num_q
    if agents(i) > 0  if sum(con(i,:)') > agents(i)  dq(i,j) = dq(i,j) + ...
                      penalty*(sum(con(i,:)'-agents);  end
  end
end
% Optimised version of the penalty function method
% dq = dq + (ones(6,1)*((agents > 0).*((sum(con') - agents) > 0)*...
% penalty.*(sum(con') - agents)))';

dq = dq(:)';
```

D.4 Code for Scaled Conjugate Gradients

This piece of code is the minimisation algorithm known as scaled conjugate gradients, which was used to carry out the resource allocation.

```
function [x, options, errlog, scalelog] = scg(f, x, options, gradf, ...
                                    P1, P2, P3, P4, P5, P6, P7, P8, P9, P10)
%SCG Scaled conjugate gradient optimization.
%
% Description
% scg('f', x, options, gradf) uses a scaled conjugate gradient
% algorithm to find the minimum of the function f(x) whose gradient is
% given by gradf(x).  Here x is a row vector and f returns a scalar
% value.   scg(f, x, options, gradf, p1, ..., p10) allows up to 10
% additional arguments to be passed to f() and gradf().     The
% optional parameters have the following interpretations.
%
% options(1) is set to 1 to display error values; also logs error
% values in the return argument errlog, and values of the scaling
% parameter in the return argument scalelog.
%
% options(2) is a measure of the precision required for the value of x
% at the solution.
%
% options(3) is a measure of the precision required of the objective
% function at the solution.  Both this and the previous condition must
% be satisfied for termination.
%
% options(9) is set to 1 to check user defined gradient function.
%
% options(14) is the maximum number of iterations; default 100.
%

% Copyright (c) Christopher M Bishop and Ian T Nabney (1996)

%  Set up the options.

if(options(14))
  niters = options(14);
else
  niters = 100;
end

options = foptions(options);

display = options(1);
gradcheck = options(9);

funcstr = [f];
funcstr=[funcstr, '(x'];
for i=1:nargin - 4
  funcstr = [funcstr,',P',int2str(i)];
end
funcstr = [funcstr, ')'];

gradstr = [gradf];
    gradstr=[gradstr, '(x'];
for i=1:nargin - 4
    gradstr = [gradstr,',P',int2str(i)];
end
gradstr = [gradstr, ')'];

%  Check gradients

nparams = length(x);

if(gradcheck)
  epsilon = 1.0e-6;
  xorig = x;
  deltaf = zeros(size(x));
  for i = 1:nparams
    x(i) = xorig(i) + epsilon;
    fplus = eval(funcstr);
    x(i) = xorig(i) - epsilon;
    fminus = eval(funcstr);
    deltaf(i) = 0.5*(fplus - fminus)/epsilon;
    x(i) = xorig(i);
  end
  gradient = eval(gradstr);
  fprintf(1, 'Checking gradient ...\n\n');
  fprintf(1, '   analytic   diffs     delta\n\n');
  disp([gradient', deltaf', gradient' - deltaf'])
end

% Main optimization loop.

errlog = zeros(1, niters);
scalelog = zeros(1, niters);

sigma0 = 1.0e-4;
fold = eval(funcstr); % Initial function value.
options(10) = options(10) + 1; % Increment function evaluation counter.
grad = eval(gradstr); % Initial gradient.
options(11) = options(11) + 1; % Increment gradient evaluation counter.
srch = - grad; % Initial search direction.
success = 1;
lambda = 1.0; % Initial scale parameter.
lambdamin = 1.0e-15;
lambdamax = 1.0e100;
n = 1; % n counts number of iterations.
nsuccess = 0; % nsuccess counts number of successes.
xval = x;

while (n <= niters)

  % Calculate first and second directional derivatives.

  if (success == 1)
    mu = srch*grad';
    if (mu >= 0)
      srch = - grad;
      mu = srch*grad';
    end
    kappa = srch*srch';
    sigma = sigma0/sqrt(kappa);
    x = xval + sigma*srch;
    gplus = eval(gradstr);
    options(11) = options(11) + 1;
    gamma = (srch*(gplus' - grad'))/sigma;
  end

  % Increase effective curvature and evaluate step size alpha.

  delta = gamma + lambda*kappa;
  if (delta <= 0)
    delta = lambda*kappa;
 lambda = lambda - gamma/kappa;
  end
  alpha = - mu/delta;

  % Calculate the comparison ratio.

  x = xval + alpha*srch;
  fnew = eval(funcstr);
  options(10) = options(10) + 1;
  rho = 2*(fnew - fold)/(alpha*mu);
  if (rho  >= 0)
    success = 1;
  else
    success = 0;
  end

  % Update the parameters to new location.

  if (success == 1)
    xval = xval + alpha*srch;
    nsuccess = nsuccess + 1;
    x = xval;
    fold = fnew;
  end

  if (display)
    errlog(n) = fold;
    scalelog(n) = lambda;
    fprintf(1, 'Cycle %4d  Error %11.6f  Scale %e\n', n, fold, lambda);
  end

  if (success == 1)
    % Test for termination

    if (max(abs(alpha*srch)) < options(2) & max(abs(fnew-fold)) < options(3)
      options(8) = fold;
      return;

    else
      gold = grad;
      grad = eval(gradstr);
      options(11) = options(11) + 1;
    end
  end

  % Adjust lambda according to comparison ratio.

  if (rho < 0.25)
    lambda = 4.0*lambda;
    if (lambda > lambdamax)
      lambda = lambdamax;
    end
  end
  if (rho > 0.75)
    lambda = 0.5*lambda;
    if (lambda < lambdamin)
      lambda = lambdamin;
    end
  end

  % Re-compute search direction using Hestenes-Steifel formula, or re-start
  % in direction of negative gradient after nparams steps.

  if (nsuccess == nparams)
    srch = -grad;
    nsuccess = 0;
  else
    if (success == 1)
      beta = (gold - grad)*grad'/mu;
      srch = - grad + beta*srch;
    end
  end
  n = n + 1;
end

% If we get here, then we haven't terminated in the given number of
% iterations.

options(8) = fold;
if (options(1) >= 0)
  disp('Warning: Maximum number of iterations has been exceeded');
end
```

D.5 Code to compute the required number of agents

The following programs evaluate the required number of agents in the system in order to achieve a proper service level.

D.5.1 Number of agents in each queue

```
function H = H_ratio(volume,time_steps_a_day)

% This function calculates the number of servers required
% if we merge all the queues(M/M/c system) in one system
% considered as a M/H6/c system
% It also computes the normalized number of agents
% required in each queue

% volume provided all the data describing the six queues:
% number of incoming calls per queue and per half-hour period
% mean service time per queue and per half-hour period
% number of server required in each queue considered separately

% time_steps_a_day is the number of half hour periods in a day

% number of queues in our system
num_q = size(volume,1)/time_steps_a_day;
% number of days considered in the week
num_day = size(volume,2)/3;
% total number of half hour periods for the whole week
num_time_steps = num_day*time_steps_a_day;

% queue contains the numbers of calls active at an instant in each queue
queue = zeros(1,num_q);

% the required number of agents in each queue for each half hour period
q = zeros(num_time_steps,num_q);
% the total number of servers, i.e. agents required in the system
c = zeros(num_time_steps,1);
% the server utilisation
ro = zeros(num_time_steps,1);

% First line: call volume in each queue
% Second line: average call duration in each queue
param = zeros(2,num_q);

for n = 1:1:num_time_steps

  total = 0;

  % Search for the call volume in the matrix volume
  for i = 0:1:num_q-1
    param(1,i+1) = volume(rem(n-1,time_steps_a_day)+1+time_steps_a_day*i,...
                          3*ceil(n/time_steps_a_day)-2);
  end
  % Search for the average call duration in the matrix volume
  for i = 0:1:num_q-1
    param(2,i+1) = volume(rem(n-1,time_steps_a_day)+1+time_steps_a_day*i,...
                          3*ceil(n/time_steps_a_day)-1);
  end

  % Count the total number of calls active at an instant
  for i = 1:1:num_q
    queue(1,i) = param(1,i)*param(2,i)/(30*60);
    total = total + queue(1,i);
  end

  % Compute the total number of agents required in the system
  % to answer 95% of all calls within 30 seconds
  if total > 0  w = zeros(1,2);
        w = H_agents(param,95,30);
        r = w(1)/total;
        c(n,1) =  w(1);
        ro(n,1) = w(2)/w(1);
   else r = 1; ro(n,1) = 1; c(n,1) = 0;
  end

  % Compute the number of agents required in each queue
  for m = 1:1:num_q
    q(n,m) = queue(1,m)*r;
  end

end

% The function returns q, c and ro
H = [ q c ro ];
```

D.5.2 Number of agents to achieve the right service level

```
function H = H_agents(param,p,t)

% H_agents calculates the minimum number of agents in order to
% answer p% of all the calls within t seconds
c = 1;
w = zeros(1,2);
w = W_q(param,c);

% The average waiting time must be less than t/log(100/(100-p))
while w(1) > (t/log(100/(100-p)))
  c = c+1;
  w = W_q(param,c);
end

n_agents = c;

% a is the offered load in erlangs of the system
a = w(2);

% the function returns n_agents and a
H = [ n_agents, a];
```

D.5.3 Average queueing time

```
function w = W_q(param,c)

% This function calculates Wq the average queueing time
% for a M/H6/c system:

% Exponentially distributed arrival times

% The service time distribution is a 6 stages hyper-exponential distribution
% (the parameters describing this two distributions are in param

% c servers

% lambda is the total arrival rate, that is equal to the sum of
% the six single arrival rates of each queue

num_q = size(param,2);

lambda = 0;
for i = 1:1:num_q
  lambda = lambda + param(1,i);
end
lambda = lambda/1800;

% E_s is the expected value of s, the service time
E_s = 0;
for i = 1:1:num_q
  E_s = E_s + param(1,i)*param(2,i)/1800;
end
E_s = E_s/lambda;

% a is the offered load of the system
a = lambda*E_s;

% the offered load must be strictly less than the number of servers
% else the number of calls waiting grows without limit
if (c-a) < 0  q = 1000000;
else
  E_s2 = 0;
  for i = 1:1:num_q
    % E_s2 is the expected value of s^2
    E_s2 = E_s2 + param(1,i)*param(2,i)^2/1800;
  end
  E_s2 = 2*E_s2/lambda;
  C = Erlang_C(c,a);
  % q is the average waiting time in the system
  q = C/(c-a)*E_s2/(2*E_s);
end

% The function returns q, the average queueing time in the system,
% and a, the offered load of the system
w = [ q a ];
```

D.5.4 Erlang *C* function

```
function C = Erlang_C(c,a)

% This function returns the value of the Erlang C function:
% c is the number of servers in the system
% a is the offered load  in erlangs of the system

% B is the Erlang B function
B = 1;

for k = 1:1:c
  B = a*B/(k+a*B);
end;

C = c*B/(c-a*(1-B))
```

D.6 Simulation

The following code simulates the activity of a call centre and returns the service level and the quality of service achieved.

```
% Simulation

% Read in the set of random numbers
load random.dat

% Read in the agents matrix
load H_ag.dat;
% Load volume information
load volume;
% time_steps_a_day is the number of half hour periods in a day
time_steps_a_day = 20;
% number of queues in the system
num_q = size(volume,1)/time_steps_a_day;
% number of days considered in the week
num_day = size(volume,2)/3;
% total number of periods for the whole week
num_time_steps = num_day*time_steps_a_day;
% num_sk is the number of skills mixes
num_sk = size(H_ag,2);

% Read in the skills matrix
load skills;
% Rescale to range 0-1.
skills = skills ./100.0;
% Only use the first num_sk skills profiles to make it interesting
skills = skills(1:num_sk, :);

% In the first column of result are stored the percentage of calls that
% exprienced a waiting time greater than 30 seconds
% In the second column are stored the values of the quality ratio
result = zeros(num_time_steps,2);

% The simulation runs throughout a whole week
for n = 1:1:num_time_steps

  % incoming_calls describes the arrivals of calls in each queue
  % for the 1800 seconds of an half hour period
  % incoming_calls(100,4)=1 means one arrival in the 100th second
  % incoming_calls(100,4)=0 means no arrival in the 100th second
  incoming_calls = zeros(num_q,1800);

  % queues contains the number of calls waiting in each queue
  queues = zeros(1,num_q);

  % total is the counter of incoming calls
  % in the queues for an half hour period
  total = zeros(1,num_q);

  % waiting_times contains the waiting times
  % of all the calls waiting in the queues
  % The first line correspond to the first arrived
  % (i.e. the longest waiting) calls
% The maximum number of calls waiting in a queue is fixed to 200
waiting_times = zeros(200,num_q);

% over_waiting counts the number of calls that experience a waiting time
% greater than 30 seconds for each queue
over_waiting = zeros(1,num_q);

% quality is the sum of the skills of the answering agents
% in an half hour period
quality = 0;

% agents contains the number of agents required in each skills mix pool
agents = H_ag(n,:);

% service_time contains the remaining time
% for each call which is being served
% service_time is initialized with -1 values
service_time = -ones(num_sk,max(agents));


% The elements of the matrix service time which could contain service time
% are initialised to 0, the others will remain equal to -1
for s = 1:1:num_sk
  service_time(s,1:agents(s)) = zeros(1,agents(s));
end

% The first line of param contains the number of incoming calls for each
% half hour period in each queue
% It is used to compute the arrival rate: lambda
for i = 0:1:num_q-1
  param(1,i+1) = volume(rem(n-1,time_steps_a_day)+1+time_steps_a_day*i,...
                 3*ceil(n/time_steps_a_day)-2);
end

% The second line of param contains the average service time for each
% half hour period in each queue
% It is used to compute the service rate: mu
for i = 0:1:num_q-1
  param(2,i+1) = volume(rem(n-1,time_steps_a_day)+1+time_steps_a_day*i,...
                        3*ceil(n/time_steps_a_day)-1);
end

% incoming_calls is filled according to this rule
% p < lambda: one arrival
% p > lambda: no arrival
for i = 1:1:num_q
  incoming_calls(i,:) = random(:,i)'<param(1,i)/1800;
end

% For each second of the half hour period
% the different parameters are computed
for i = 1:1:1800

  % We add to the current queues the new arrivals
  queues = queues + incoming_calls(:,i)';
```

```
% We add to the total number of incoming calls the new ones
total = total + incoming_calls(:,i)';

% When the service time is completed the agent become available
% So we add him to the counter of free agents
for s = 1:1:num_sk
  agents(s) = agents(s) + sum(service_time(s,:) == 1);
end

% We decrease the remaining service time by one second
for s = 1:1:num_sk
  service_time(s,find(service_time(s,:) > 0)) = ...
               service_time(s,find(service_time(s,:) > 0)) - 1;
end

% For each queue
for j = 1:1:6

  m = 0;
  exit = 0;

  % While there is at least one call in one of the queues and
  % While there is at least one agent qualified to answer that call
  while queues(j) > 0 & exit == 0;
    % available contains the skills of the avaible agents
    available = (agents ~= 0).*skills(:,j)';
    % If there is one agent qualified to answer the calls from queue j
    if max(available) > 0
      % add to quality the skill of the most qualified agent available
      quality = quality + max(available);
      % m is the number of the pool whose agents are
      % the most qualified to answer that call
      [ Y I ] =  sort(agents.*(available == max(available)));
      m = I(num_sk);
      % We decrease by one the number of agents in the pool number m
      agents(m) = agents(m) - 1;
      % We decrease by one the number of calls waiting in the queue j
      queues(j)  = queues(j) - 1;
      % We compute the service time for answering that call
      % which is taken randomly from an exponential distribution
      service_time(m,min(find(service_time(m,:) == 0))) =...
      ceil(param(2,j)*log(1/(1-random(i,7))));
      % When a call is answered, the next waiting call becomes the
      % first which will be answered
      for k = 1:1:199
        waiting_times(k,j) = waiting_times(k+1,j);
      end
      waiting_times(200,j) = 0;
    else
      % If there is no agent available to answer
      % the calls from queue j we quit the loop
      exit = 1;
    end
  end
```

```
      % The waiting times of the waiting calls are increased by one
      if queues(j) ~=0  waiting_times(1:queues(j),j) = ...
                                 waiting_times(1:queues(j),j) + 1;
      end

      % If a waiting_time becomes greater than 30, it is counted
      over_waiting(j) = over_waiting(j) + sum(waiting_times(:,j)==31);

    end

  end

  % Print the percentage of calls that have experienced
  % a waiting time greater than 30 seconds
  result(n,1) = sum(over_waiting)/sum(total);

  % Print the value of the quality ratio for the current half hour period
  result(n,2) = quality/sum(total);

end

save 'result2.dat' result -ascii
```

Bibliography

[1] D. Lowe and I. Nabney. Human resource allocation in a call centre: Feasibilty study report, August 1996.

[2] A. O. Allen. *Probability, Statistics and Queueing Theory with computer science applications.* Academic Press, Inc., 1978.

[3] L. Kleinrock. *Queueing Systems Volume I: Theory.* John Wiley and Sons Inc., 1975.

[4] L. Kleinrock. *Queueing Systems Volume II: Computer Applications.* John Wiley and Sons Inc., 1976.

[5] P. Beckmann. *Introduction to Elementary Queueing Theory and Telephone Traffic.* The Golem Press, 1968.

[6] A. M. Lee. *Applied Queueing Theory.* Mac Millan, 1966.

[7] T. L. Saaty. *Elements of Queueing Theory with Applications.* McGraw-Hill Book Company, 1961.

[8] C. M. Bishop. *Neural Networks and Pattern Recognition.* Oxford University Press, 1995.

[9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1992.

[10] *MAXcaster User's Manual.* Erlang Calculations.