MASTER'S THESIS

Workforce Distribution in Dynamic Multi-Agent Systems

by

David Millán Ruiz

A Dissertation Presented to the Department of Artificial Intelligence

of the

National Distance University of Spain

In Partial Fulfilment of the Requirements for the Degree of

Master of Advanced Artificial Intelligence

Advisor: Severino Fernández Galán

Madrid, Spain
July 2010

Dedicated to my parents and my beloved Fanny Michel

Index

DECLARATION OF AUTHORSHIP

The author, David Millán Ruiz, hereby declares and confirms that this dissertation is entirely the result of the work carried out in the Department of Artificial Intelligence of the School of Informatics at the National Distance University of Spain (UNED). This dissertation contains original contribution by the author unless otherwise indicated.

David Millán Ruiz

July 2010

ABSTRACT

This work describes a novel approach to workforce distribution in dynamic multi-agent systems based on backboard architectures. These environments entail quick adaptations to a changing environment that only fast greedy heuristics can handle. These greedy heuristics consist of a continuous re-planning, considering the current state of the system. As these decisions are greedily taken, the workforce distribution may be poor for middle and/or long term planning due to incessant wrong movements. The use of parallel memetic algorithms, which are more complex than classical, ad-hoc heuristics, can guide us towards more accurate solutions. In order to apply parallel memetic algorithms to such a dynamic environment, we propose a reformulation of the traditional problem, which combines predictions of future situations with a precise search mechanism, by enlarging or diminishing the time-frame considered. The size of the time-frame depends upon the dynamism of the system (smaller when there is high dynamism and larger when there is low dynamism). This work demonstrates how nearly optimal solutions each *v* seconds (size of the time-frame) outperforms continuous bad distributions when the right size of the time-frame is determined, and predictions and optimisations are properly carried out. Specifically, we propose a neural network for predicting future system variables and a parallel memetic algorithm to perform the assignment of incoming tasks to the right agents, which outperforms other conventional approaches. Additionally, we propose a modification of the resilient back-propagation algorithm and evolutionary operators based on meta-heuristics. To conclude, we test out our method on a real-world production environment from Telefónica which is a large multinational telephone operator.

ACKNOWLEDGEMENTS

This dissertation has been carried out in the Department of Artificial Intelligence of the School of Informatics at the National Distance University of Spain (UNED), under the supervision of Prof. Dr. Severino F. Galán.

Firstly, I would like to thank Prof. Dr. Severino F. Galán for giving me the opportunity to perform this research under his supervision. He supported me and guided me from the beginning of this work, giving me many valuable and useful comments to progress in my research.

Secondly, I would also like to thank the Department of Customer Care of Telefónica Research & Development, headed by Javier Bonastre, for their support to this work and for the infrastructure provided (mainly for the Sun Fire E4900 server). Namely, Jorge Pacheco, José Luis Vélez and Jesús Celis deserve a special mention for their effort in helping me out to adapt my models and algorithms to the systems of Telefónica.

Thirdly, I would like to thank Prof. Dr. Mark Jerrum from the University of Edinburgh because he was the "seed" of all this work. He gave me the first guidelines for some of my models when I did my dissertation at the University of Edinburgh during the academic year 2005-2006.

Last but not least, I also want to thank the reading committee, for the time spent in carefully reviewing this work.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ACD	Automatic call distributor
ANN	Artificial neural network
AI	Artificial intelligence
ARIMA	Autoregressive integrated moving average
ARPU	Average revenue per user
ARR	Absolute right rate
BB	Branch and bound
BPN	Back-propagation
CC	Call centre
CG	Call group
COP	Constraint optimisation problem
DMAS	Dynamic multi-agent system
DP	Dynamic programming
DTTS	Dumped trend time series
EA	Evolutionary algorithm
EC	Evolutionary computation
ES	Exponential smoothing
GA	Genetic algorithm
GRASP	Greedy randomised adaptive search
GS	Global search
GWB	Greedy workload balancing
HGRASP	Hybrid greedy randomised adaptive search
IA	Intelligent agent
ILS	Iterated local search
LR	Lineal regression
LS	Local search
MA	Memetic algorithm
MAE	Mean absolute error
MAPE	Mean absolute percentage error
MAS	Multi-agent system
MOGA	Multi-objective genetic algorithm
MSS	Multi-start search
MSCC	Multi-skill call centre
MSE	Mean square error
NSGA	Non-dominated sorting genetic algorithm
PCA	Principal component analysis
PD	Poisson distribution
PRR	Percentage of right rate
RM	Regression model

Rprop	Resilient back-propagation
RR	Right rate
RWB	Random workload balancing
SA	Simulated annealing
SBR	Skill-based routing
SES	Simple time series
SSE	Sum squared error
STS	Stationary time series
TaS	Tabu search
TS	Time series
uRprop	Upgraded resilient back-propagation
VND	Variable neighbourhood descent
VNS	Variable neighbourhood search
WCOP	Weighted constraint optimisation problem

CHAPTER 1. INTRODUCTION

1.1 OVERVIEW

Over the last years, a gradually-growing interest in parallel and distributed computing has arisen in computer science. This concern has guided research activities to areas such as parallel and distributed programming, distributed information systems, and parallel and distributed hardware architectures. Truthfully, there exists a vast bibliography (e.g. see [1-3]) about this issue, although there are still paths to explore.

Furthermore, we perceive a tendency to tackle increasingly complex problems and application domains which frequently involve the processing of continuous, dynamic data flows. These arduous environments are usually hard to be efficiently maintained by conventional and sequential techniques. Nevertheless, parallel and distributed methods not only mitigate this drawback but also present several valuable characteristics such as robustness, traceability, problem simplification, adaptivity, scalability and speed-up.

Conversely, dynamics, synchronisation and behaviour appear as intricacies of parallel and distributed information systems because the representation of linear problems into sub-problems is not always feasible or straightforward.

Anyhow, parallel and distributed systems should somehow self-improve to attain high performance. In fact, nowadays, a wide range of studies on adaptive techniques in parallel and distributed information systems can be found [4, 5].

A classical, well-suited problem for studying dynamic systems is the workload distribution in multi-agent systems. Agents can work for a common goal, coordinate the plans or draw up a plan for others' tasks. Although there are lots of multi-agents systems, we will focus on those encapsulated in blackboard architectures [6, 7]. In other words, we will work on systems with a common repository of knowledge.

The basic variant of a workforce distribution problem requires the assignment of task to agents who have the required skills to handle them over time, satisfying a given set of additional constraints and respecting the dependencies among individual tasks and differences in the execution skills of the agents. This problem has multiple variants but, depending on the dynamism of the system, we can principally distinguish two main scenarios:

1) On the one hand, we can find short-term planning environments in which a continuous planning is needed due to the high dynamism of the system. These solutions attempt to distribute the workload among agents by applying "basic" ad-hoc heuristics, looking at the current situation (without predictions or predictions for a short time-frame). This feature can be effortlessly seen in workload allocation within a *dynamic multi-skill call centre* [8].
2) On the other hand, we can find long-term planning systems in which the list of tasks is predefined and known by all agents like in the *classic scheduling problem* [9]; or environments in which a single task type is assigned to each agent for a long period of time, similarly to the *job assignment problem* [10]. In other cases, agents are assigned to patterns of tasks, instead of specific tasks (such as in *pattern-based scheduling* [9]). Analogously, stable multi-skill call centres [8] can be also included in this group. These solutions consider stable behaviour over time, anchored in historical data and apply more complex algorithms to match agents and task types. However, when having a dynamic system, these approaches cannot be efficiently applied, since an adaptive method is required.

Our proposal is encapsulated in the first scenario: dynamic systems. We put forward an alternative approach to traditional solutions which relies on an adaptive middle-term time-frame, instead of a short-term one (when the dynamism is very low, it is analogous to having a long time-frame). In other words, we reformulate the traditional problem by dynamically enlarging or diminishing the time-frame considered to better adapt the algorithm to the current state of the system. Figure 1 explains where our approach is positioned. Besides, we provide the required mechanisms to implement this more efficient, adaptive solution. Although this solution can be extended to countless domains and multi-agent systems, we will go over the call centre application (see Chapter 5) in order to examine its idiosyncrasy and complexity.

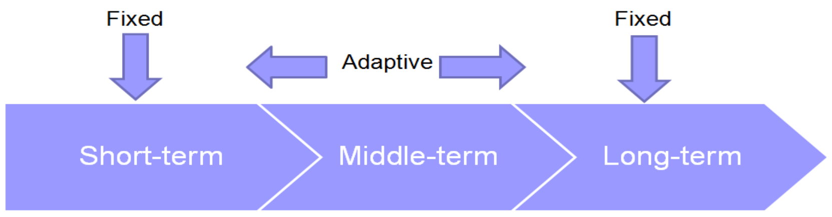


Figure 1: Adaptive time-frame.

Table 1 summarises some fundamental characteristics of the previously described scenarios in relation to the time-frame considered.

Table 1: Comparison of the time-frame considered for the workforce distribution problem.

Time-frame	*Complexity*	*Response time*	*Adaptability*	*Performance*	*CPU Utilisation*
Short-term planning	low	low	medium	medium	low
Middle-term planning	high	medium	high	high	high
Long-term planning	medium	high	low	low	high

To conclude this outline, we would like to stress that this study has been applied to (and supported by) Telefónica (http://www.telefonica.com). Telefónica is one of the world's largest telecommunications companies by market capital. Its activities are mainly centred on the fixed and mobile telephony businesses, while its broadband business is the key growth driver, underpinning both. It operates in *25* countries and its customer base exceeds *264* million people worldwide. Telefónica's growth strategy is focused on the markets in which it has a strong foothold: Spain, Europe and Latin America. The Group stands in third position in the sector Telco worldwide in terms of market capitalisation, the 1st as an European integrated operator and also the third in the Eurostoxx *50* ranking, composed of the major companies in Europe (December 31st 2009).

1.2 MOTIVATION AND MAIN OBJECTIVES

The problem of workload distribution in multi-agent systems is an appealing and challenging subject of research not only from the point of view of machine learning but also from a business angle. The eminent complexity of this problem makes it even more interesting and a firm member of the class of NP-hard problems [13]. Besides, timing constraints complicate, even more, finding an accurate, feasible solution. Another reason to analyse this problem is that it is often omnipresent in our daily life and is highly relevant to many industrial application domains like trading and workflow organisation.

From a parallel computing angle, this problem is also tempting since it inherently allows for parallelism because the tasks to handle can be distributed over several nodes and also because the nodes can execute different tasks in parallel.

From an artificial intelligence point of view, this problem is also very motivating because it involves many fields which range from forecasting techniques derived from machine learning theory to optimisation algorithms that use diversity maintenance techniques from evolutionary computation (EC) and other local search schemes like simulated annealing or tabu search.

The main purpose of this work is to provide a solution, which is fully described in Chapter 4, for dynamic multi-agent systems based on blackboard architectures. Thus, an efficient forecasting method must be provided in order to predict the real situation in next time-frame (future system state) and, therefore, an optimisation algorithm must be performed to determine the right assignment *task-agent*.

1.3 CONTRIBUTIONS

The contributions of this work can be devised from diverse perspectives although the main contribution is the presentation of a novel approach to the workforce distribution problem which coalesces forecasting with optimisation by considering an adaptive middle time-frame. We also apply this approach to a real-world production environment (multi-skill call centre) from one of the largest telecom operators around the world (Telefónica).

Typically, traditional process management systems rigidly distribute tasks to queues from which agents take and process work, regularly opting for the precise tasks they actually desire to cope with. In contrast, our approach enhances workforce distribution by additionally injecting real-time knowledge of the task, individual skill sets, and availability and utilisation of the workforce, allowing for dynamic and active distribution of tasks over time.

Additionally, our method provides further clearness on customer service level agreements and endows with insights into optimisation, offering outstanding customer service.

In addition, our approach enables us to work at a lower level of granularity (fine-grain) than short-term algorithms do (coarse-grain), because our search algorithm has more time to find a solution than conventional techniques, thanks to the predictions of future states. We can then work at agent's profile level instead of predefined sets of agents as other methods impose. Other conventional techniques consider steady environments which are far from the soundness of a dynamic mechanism.

Furthermore, other technical contributions of this dissertation can be summarised as follows:

1) This work proposes a parallelisable approach based on island models to a real-world NP-hard problem, using different fields from Artificial Intelligence.
2) New genetic algorithm operators are proposed in order to maintain a balance between diversity and intensity when searching in such an environment. These operators are often inspired in other meta-heuristics schemes.
3) We also propose a partial fitness function in order to speed-up the evaluations of candidate solutions.

4) Three exhaustive comparisons among different classical forecasting techniques, various classical heuristics for dynamic multi-agent systems and other meta-heuristics applicable to dynamic multi-agent systems are provided from multiple points of view.

Finally, the contributions to the scientific literature have produced the following peer-reviewed publications ((1) and (2) are less directly related to this dissertation):

1) Martínez-López, R.; Millán-Ruiz, D.; Martín-Domínguez, A. and Toro-Escudero, M.A.: *An Architecture for Next-Generation of Telecare Systems Using Ontologies, Rules Engines and Data Mining*. Proceedings of the International Conferences on Computational Intelligence for Modelling, Control and Automation; Intelligent Agents, Web Technologies and Internet Commerce; and Innovation in Software Engineering (CIMCA 2008), p. 31-36, Vienna, Austria, December 10-12, 2008.
2) Melendez, J.; López, B. and Millán-Ruiz, D.: *Probabilistic models to assist maintenance of multiple instruments*. Proceedings of the 14th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2009), p. 1499-1503, Palma de Mallorca, Spain, September 22-26th, 2009.
3) Pacheco, J.; Millán-Ruiz, D. y Vélez, J.L.: *Neural Networks for Forecasting in a Multi-skill Call Centre*. Proceedings of the 11th International Conference on Engineering Applications of Neural Networks (EANN 2009), p. 291-300, London, UK, August 27-29, 2009.
4) Millán-Ruiz, D. and Hidalgo, I.: *A Memetic Algorithm for Workforce Distribution in Dynamic Multi-Skil Call Centres*. Proceedings of the 10th European Conference on Evolutionary Computation in Combinatorial Optimisation (EVOCOP 2010), p. 178-189, Istanbul, Turkey, April 7-9, 2010.
5) Millán-Ruiz, D.; Pacheco, J.; Hidalgo, I. y Vélez, J.L.: *Forecasting in a Multi-skill Call Centre*. Proceedings of the 10th International Conference on Artificial Intelligence and Soft Computing (ICAISC 2010), Zakopane, Poland, June 13-17, 2010.
6) Millán-Ruiz, D. and Hidalgo, I.: *Algoritmo memético paralelo para la distribución de esfuerzo en centros de llamadas dinámicos multiagente y multitarea*. (Accepted) To appear in the 7th Spanish Conference on Meta-

heuristics, Evolutionary Algorithms and Bioinspired Algorithms (MAEB 2010), Valencia, Spain, September, 2010.

7) Millán-Ruiz, D. and Hidalgo, I.: *Comparison of Metaheuristics for Workforce Distribution in Multi-Skill Call Centres*. Submitted to the International Joint Conference on Computational Intelligence (ICEC 2010).

8) Millán-Ruiz, D. and Hidalgo, I.: *A Self-Tuning Hybrid Memetic Algorithm for Dynamic Multi-Agent Systems based on Blackboard Architectures*. Submitted to the Workshop on Self-tuning, self-configuring and self-generating search heuristics (Self* 2010). Extended versions of selected contributions from this workshop will be considered for publication in a Special Issue of the Evolutionary Computation Journal, MIT Press.

1.4 MARKET RELEVANCE

The market relevance of the present work can be devised from three distinct angles: *customer satisfaction* (happy customers remain loyal to their telecom operator), *optimisation of resources* (monetary savings as a result of a better workforce distribution) and *employee satisfaction* (brand pride, self-esteemed fortification and fair workload allocation).

Various studies [14, 15] prove that users' key period to migrate to another telecom operator (also denoted as *churn*) after having a negative experience with call centre's (CC) contact service is, for the majority of individuals, about *10* days from the notification date. During these days, and even afterwards, these people negatively influence their social circles or communities, causing a cascade effect which implies huge losses of money to telecom operators every year.

If an organisation is planning to link up with any existing outbound churn prediction models, based on likelihood to churn, then there appears the need of having a process for risk prioritisation (potential churners are prioritised) built, if the company has capacity constraints in its outbound CC (when there are not enough resources to directly contact potential churners, an automatic risk prioritisation in needed in the inbound CC).

However, we should highlight we cannot evade churn effect as far as it decidedly depends upon multiple individual reasons. There, CCs can play an important role in churn prevention as a consequence of customer satisfaction enhancement which irrefutably leads us to customer loyalty.

By upgrading customer service with our approach, Telefónica has estimated savings of up to *€2,000,000* per year only in Spain, as it enhances brand loyalty (customers are happier with their telecom operator) and other encouraging behaviours such as word of mouth advocacy.

Nevertheless, customer satisfaction is not the unique edge from where we can profit. Another important aspect refers to the optimisation of resources we are actually doing because we increase the speaking level of each agent. If we consider the mean upgrading percentage obtained by our approach in 2009 (*7%*), we can affirm that, only in Spain, it is possible to obtain savings up to *€3,000,000* per year (Figure 2 shows the two main bases of success for a telecom operator).

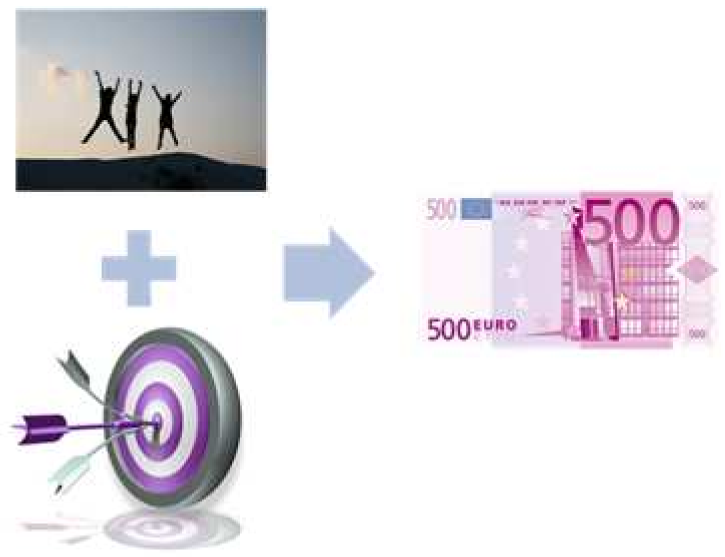


Figure 2: Customer satisfaction and resources optimisation can be monetised.

Besides, we can enhance another important factor; the employee satisfaction. Maintaining morale high among agents can be of extraordinary benefit to any telecom operator, as happy agents will be more prone to reply to more incoming calls and stay loyal to the company. This occurs because agents are much fairly treated as a consequence of a better allocation of workload. Agents cannot feel they are being fairly treated whether other agents have to work less time, earning the same money and having similar (or even identical) skills.

As a final point, we can extend this work to many other dynamic multi-agent systems in which the list of tasks is not predefined such as plane maintenance [16], online trading [17], disaster response [18], congestion in stations [19] or overloading in networking nodes [20].

1.5 DISSERTATION ORGANISATION

The rest of this document is organised as indicated in the present section.

Chapter 2 introduces the problem of workload distribution in dynamic multi-agent systems from a generic point of view in Section 2.1 and from formal perspective in Section 2.2. In Section 2.3, we address the difficulty of handling hard and soft constraints.

Chapter 3 presents an overview of those aspects of research that are relevant to the problem faced. The required background for situating the work presented in this document and making a proper understanding of it is given in Section 3.1. Section 3.2 presents a survey of existing work from different points of view, considering commonalities with other problem domains. Section 3.3 kindly discusses the state-of-the-art and introduces some bases to outperform related work.

Chapter 4 proposes a new approach to the problem addressed in this work. Section 4.1 sets out the bases of this novel approach. Section 4.2 describes the methodology that we have followed. Section 4.3 focuses on the forecast component which supports the first module of our approach. Section 4.4 describes the second component of our approach; in other words, the search module. In Section 4.4, we also propose multiple mechanisms to maintaining a fair balance between diversity and intensity in simple and parallel genetic algorithms when optimising.

In Chapter 5, we adapt our approach to a real-world DMAS: the multi-skill call centre. Section 5.1 describes the specific characteristics of our problem domain. Section 5.2 presents a brief survey of call centre algorithms. Section 5.3 highlights the magnitude, in terms of volume, of our application domain. In Section 5.4, we present some special adaptations for the forecast module. In contrast, Section 5.5 points out some particular adaptations for the search module.

Section 6.1 describes the dataset employed. Section 6.2 points out the hardware descriptions of the SunFire sever in which the evaluations have been performed. Section 6.3 analyses the selected metrics for testing and comparing our approach. Section 6.4 examines the forecast module for five different CGs as there are too many to accomplish an exhaustive study for all of them. Section 6.5 evaluates the search module by studying several time intervals from days with different complexity; this section also compares our search module with other acknowledged techniques. In Section 6.6, we will analyse our complete approach (forecast module + search module) for one-day campaign. We will also compare how our complete approach outperforms other conventional call centre’s algorithms.

Chapter 7 concludes our work with a summary of major contributions in Section 7.1 and points out prospects for future work in Section7.2.

CHAPTER 2. PROBLEM OF WORKLOAD DISTRIBUTION IN DYNAMIC MULTI-AGENT SYSTEMS

The present chapter describes the problem of workload distribution in dynamic multi-agent systems from different perspectives. The main aim of this chapter is to introduce this problem to the reader as the pillars of the present work rely on the concepts given throughout the following sections.

Section 2.1 presents the problem of workload distribution in dynamic multi-agent systems from a generic point of view. Section 2.2 formalises the problem definition in order to provide the present work with a higher level of scientific rigor. In Section 2.3, we tackle the difficulty of handling hard and soft constraints as it is the typical situation in real-world environments.

2.1 GENERIC DEFINITION OF THE PROBLEM

The term *intelligent agent* (IA) [21] describes an autonomous entity which is able to observe and interact with its environment in order to accomplish a given set of tasks [22]. IAs may also learn from their environment or use previous knowledge of the domain to achieve their goals. Their complexity can range from very simple systems to very complex ones. Unlike objects, which are defined in terms of methods and attributes, an agent is defined in terms of its behaviour.

Different authors [21-23] have proposed diverse definitions of agents which commonly include concepts such as *persistence* (code is not executed on demand and decides for itself when it should perform a given activity), *veracity* (an agent cannot communicate false information), *kindness* (agents do not have conflicting goals), *rationality* (agents will act in order to achieve their goals), learning (agents improve performance over time), *autonomy* (agents have capabilities of task selection, prioritisation and goal-oriented behaviour), *sociability* (agents are able to engage other components through some sort of communication and coordination, so that they may collaborate on a task) and *reactivity* (agents perceive the context in which they operate and react to it appropriately).

When several agents interact, these may compile a multi-agent system (MAS) [24]. Characteristically, such agents have a partial point of view of the problem and thus need to cooperate with other agents. Furthermore, there may be no global control and thus such systems are sometimes denoted as swarm systems. In these cases, data are decentralised and execution is asynchronous.

The real world is actually a multi-agent environment because we often need to cooperate with others in order to achieve our own goals. In fact, many goals can be only achieved with the cooperation of others. Social ability in agents is the ability to interact with other agents (and possibly humans) via some kind of agent-communication language.

Commonly, the basic variant of the workload distribution problem in a dynamic multi-agent system (DMAS) requires the assignment of task to agents which have the required skills to handle them over time, satisfying a predefined set of additional constraints and respecting the dependencies among individual tasks and differences in the execution skills of the agents.

In a common DMAS, there are n tasks or work items grouped in k types of tasks and m agents that may have up to l skills ($l \leq k$) to perform these works. In this manner, each agent can process different types of tasks and, given a type of task, it can be carried out by several agents that have that skill. The set of skills an agent has is frequently denoted as *profile*. These profiles can be truly heterogeneous as there are massive potential skills.

Although agents may have multiple skills, each agent can only process one operation at the same time. Furthermore, given an operation, it requires an unknown amount of time to be accomplished. Besides, each agent must orderly process each operation during an uninterrupted period of time; in other words, the task cannot be divided or postponed once it has already started.

Constraints may be given by many factors that we cannot cover in this section as this issue is problem dependent. However, we will describe how we propose to treat them in Section 2.3 and present a real example in Chapter 5.

The solution to the problem of workload distribution in dynamic multi-agent systems consists in dynamically assigning every task (according to its type) to the right agent so that this solution satisfies all hard constraints and respects, if possible, all soft constraints.

Eventually, we need a metric of quality to measure the rightness of each solution. Of course, the definition of the quality function is problem dependent too. In Chapter 5, we will show an example of quality function for the dynamic multi-skill call centre use case.

2.2 FORMAL DEFINITION OF THE PROBLEM

Formalising the definition given in Section 2.1, we can find the following parameters in a dynamic multi-agent system based on blackboard architecture:

1) a finite set of *n* tasks or work items $W = \{ w_1, w_2, \dots, w_n \}$.
2) a finite set of *k* task types $T = \{ t_1, t_2, ..., t_k \}$, where $k \le n$ when every task type has, at least, one task assigned.
3) a finite set of *m* agents $A = \{ a_1, a_2, ..., a_m \}$.
4) a finite set of *k* agent-skills $S = \{ s_1, s_2, ..., s_k \}$ in which each agent-skill, s_i, represents the capability to handle the corresponding type of task, t_i, with the equivalent sub-index in *T*: $s_1 \sim t_1, s_2 \sim t_2, ..., s_k \sim t_k$.
5) a finite set of *d* agent-skill profiles $P = \{ P_1, P_2, ..., P_d \}$ in which each agent-skill profile P_i can be any subset of $S = \{ s_1, s_2, ..., s_k \}$.
6) a finite set of *n* operations (execution or processing of each task, w_i) $O = \{ o_1, o_2, ..., o_n \}$ in which each operation, o_i, has associated a processing time which depends on its type of task: $\{\tau_1, \tau_2, ..., \tau_k\}$.

The goal is to obtain the right assignment (solution) for every agent a_i to the most suitable profile P_j from the potential skill profiles of each agent a_i for each *v* seconds, where *v* is the size of the time-frame considered.

Figure 3 illustrates a feasible solution for a given time-frame, supposing that agent a_1 has the skills to process t_1 and t_2 (s_1 and s_2), agent a_2 has the skills to process t_1 and t_k (s_1 and s_k), agent a_3 has the skill to process t_2 (s_2) and agent a_m has the skills to process t_1 and t_k (s_1 and s_k).

In addition, the assignment $\langle a_i, P_j \rangle_t$ must satisfy all hard constraints and handle the soft ones. To determine whether (or not) a given solution is appropriate, we need to define a quality metric to evaluate the rightness of each feasible solution. This function is intuitively problem dependent as aforementioned.

Moreover, the solution must fulfil the following descriptions:

1) on *O* define *R*, a binary relation which represents the precedence among operations. If $(o_1, o_2) \in R$ then o_1 has to be performed before o_2.

2) every agent, a_i, has associated a finite non-null subset of P, containing his or her skills to handle different types of tasks (individual skill-profile).
3) the same profile P_i can be assigned to several agents. In other words, several agents may have some skills in common (or even all of them).
4) every agent, a_i, may have several profiles assigned but only one can be performed at a given instant *t*, $\langle a_i, P_j \rangle_t$. Therefore, an agent cannot process two (or more) operations at the same instant.
5) every solution must respect diverse (hard and soft) constraints given by business rules defined.

Figure 3 illustrates the situation described above in this section. We present an example in which each agent has certain potential skills (at least one) to attend some tasks types. The fact that a given agent has multiple skills does not mean he must attend all these types at the same time within a given interval (do not confuse potential skills with currently assigned skills).

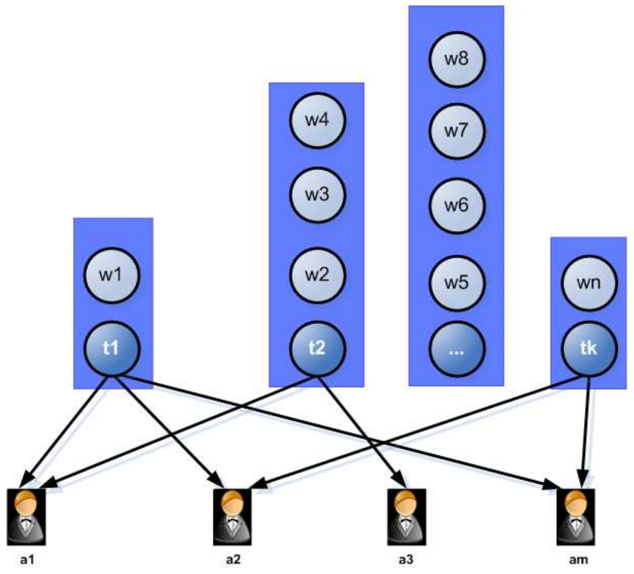


Figure 3: Multi-agent system configuration based on the potential skills of all agents.

2.3 CONSTRAINT OPTIMISATION PROBLEM

The problem described in Section 2.2 can be viewed as a constraint optimisation problem (COP) [25]. A COP is characterised by a set of v variables, $\{X_1, X_2, \dots, X_v\}$ and a set of c constraints $\{C_1, C_2, \dots, C_c\}$ for a nonempty domain D_i of feasible values.

A system state is defined by an assignment of values to some (or all) variables. An assignment that does not violate any constraints is denoted as *consistent* or *legal assignment*. A *complete assignment* is one in which every variable is mentioned, and a solution to a COP is a complete assignment that satisfies all the constraints. In our case, the constraints are associated to the tasks, the agents, timing, actions or desired/undesired situations.

Classic COPs treat constraints as hard, referring to the fact that each feasible solution must satisfy all constraints. In other words, a solution is feasible only if it satisfies every single constraint. In contrast, flexible COPs relax this assumption by partially relaxing constraints and allowing the solution not to comply with all them (soft constraints).

We consider the weighted constraint optimisation problem (WCOP) in the present work, in which each violation of a soft constraint is weighted according to a predefined relevance (relevance is usually given by the business units of a company). Consequently, satisfying soft constraints with more weight is preferred whereas hard constraints cannot be violated in any case. The violation of soft constraints is penalised according to the degree of non-accomplishment of these constraints and their relevance.

Weights can be assigned by defining level of constraints. For each level, we can define a range for the weights (constraint relevance) and the gap between two levels follows a logarithmic function in order to soften the difference among levels. Different levels cannot have the same relevance (no overlapping constraints levels) and determining the difference among levels is frequently a business driven action according to the market relevance. The values for a given level should be proportionally assigned.

Figure 4 illustrates the relationship among constraints and constraint levels for *4* levels and *7* constraints. In our example, *Level 4* (the most relevant constraint level) has two constraints where *Constraint 1* (*C_1*) has a higher weight than *Constraint 2* (*C_2*). *Level 4*'s weights range from *Ln(4)=1.386* to *Ln(4+1)=1.609*, *(1.386, 1.609]*.

Therefore, C_1 may have a weight of $1.550 \in (1.386, 1.609]$ while C_2 may have weight of $1.450 \in (1.386, 1.609]$ (these weights are fictitious, we just want to remark that C_1 has a higher weight than C_2 in Figure 4.

Level 3's weights range from *Ln(3)=1.098* to *Ln(3+1)=1.386*, *(1.098, 1.386]* and *Level 2*'s weights range from *Ln(2)=0.693* to *Ln(2+1)=1.098*, *(0.693, 1.098]*. *Level 3* and *Level 2* have a unique constraint which must belong to its respective constraint level's range. Of course, *Level 3* has more relevance than *Level 2* which has more importance than *Level 1* at the same time.

Level 1's weights range from *0.1* (we will consider *0.1* as a minimum) to *Ln(1+1)= 0.693*, *(0.1, 0.693]*. In *Level 1*, all constraints have the same relevance as take the same "space" in the level (let's say *0.5*).

Note that if we need to set up higher differences among levels, we just need to assign a higher range of weights for each level but this is problem dependent (we have just shown an example).

Finally, we need to normalise all penalisations by dividing by the total sum of weights assigned to the soft constraints.

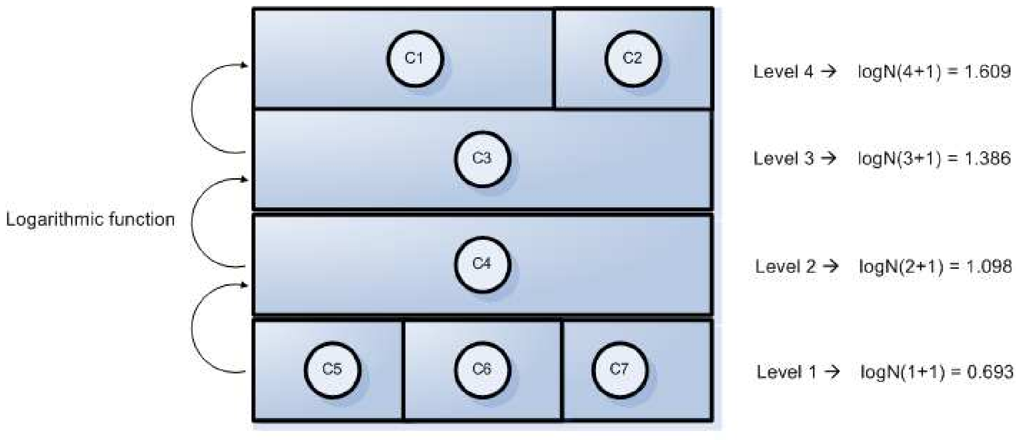

C1
C2
C3
C4
C5
C6
C7
Logarithmic function
Level 4 → logN(4+1) = 1.609
Level 3 → logN(3+1) = 1.386
Level 2 → logN(2+1) = 1.098
Level 1 → logN(1+1) = 0.693

Figure 4: Relationship among constraints.

The aim is to find a solution to the problem whose cost, evaluated as the sum of the cost functions (penalisations of soft constraints), is minimised.

CHAPTER 3. LITERATURE REVIEW

This chapter provides the reader with an overview of all relevant aspects of scientific research for the problem described in Chapter 2. The core background for situating the present work is given in Section 3.1. Section 3.2 presents a survey of related work from diverse perspectives, considering commonalities with other problem domains. In future chapters, we will examine how to adapt some of these techniques to our use case (call centre). Section 3.3 kindly discusses the state-of-the-art and introduces some bases to outperform existing work.

3.1 BACKGROUND

Throughout the present work, very heterogeneous fields from artificial intelligence (AI) have been applied. Prior to analysing existing work and proposing a novel approach to the problem of workload distribution in DMAS, it is necessary to endow the user with some required background in order to better understand diverse concepts and proposed solutions.

3.1.1 Local Search

In computer science, local search (LS) [26] is a meta-heuristic (MH) for solving computationally hard optimisation problems. LS can be pertained to problems that can be formulated as finding a solution by maximising or minimising a criterion within a set of candidate solutions.

Frequently, the neighbourhood is composed by more than one solution where the choice of which one to move to is taken by only considering information concerning the solutions within the neighbourhood of the current one. When we select a neighbour solution taking the one which maximises the criterion, then the MH is named *hill climbing*.

LS based algorithms “navigate” the search space, jumping from a solution to another one until a solution deemed optimal is reached or a given computing time has been elapsed. Another common choice is to terminate when the best solution found by the algorithm has not been improved in a given number of steps. LS algorithms are typically incomplete algorithms, as the search may stop even if the best solution found by the algorithm is not optimal. This can happen even if termination is due to the impossibility of improving the solution, as the optimal solution can lie far from the neighbourhood of the solutions crossed by the algorithms.

LS algorithms have been extensively applied to numerous hard computational problems, including problems from computer science, mathematics, operations research, engineering and bioinformatics [27].

To conclude, we provide the pseudo-code adapted to the problem of the workload distribution in dynamic multi-agent systems, which illustrates the LS algorithm in its basic form:

```
void Local_Search (Chromosome & candidate_solution)
{
 Chromosome best_solution = candidate_solution;
 Chromosome neighbour = candidate_solution;
 For (i=0; i<candidate_solution.size(); i++)
 {
     Agent a = neighbour.getAgent(i);
     For (j=0; j<a.get_number_profiles(); j++)
     {
          neighbour.change_profile(i,j);//profile j for agent i
          If (neighbour.fitness() > best_solution.fitness())
      best_solution = neighbour;
     }
     neighbour = best_solution;
 }
 candidate_solution = best_solution;
}
```

Algorithm 1: Basic LS pseudo-code.

3.1.2 Genetic Algorithms

A genetic algorithm (GA) [28, 29] is a class of adaptive stochastic optimisation techniques which attempts to find exact or approximate solutions for optimisation and search problems. GAs were proposed by John Holland in 1975 [28]. GAs are also a particular class of evolutionary algorithms (EA) that use techniques derived from evolutionary ideas of natural selection and genetics such as inheritance, mutation, selection, and crossover or recombination.

GAs are implemented in a computer simulation in which a population of abstract representations (called chromosomes or the genotype of the genome) of candidate solutions (called individuals or phenotypes) to an optimisation problem evolves toward more accurate solutions.

The evolution typically begins with an initial population of randomly generated individuals and occurs over time by means of generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are selected from the current population (based on either their fitness or composition), and modified (recombined and randomly mutated) to compose a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm ends up when a given number of generations has been produced, or after a period of time, or after *x* generations without evolution, or a satisfactory fitness level has been accomplished for the population. If the algorithm has ended up due to a maximum number of generations, a satisfactory solution may or not have been reached. Further description is given in Section 4.4, since GAs are the bases of the solution proposed in this work.

```
Procedure Basic_Genetic_Algorithm
{
    Generate an initial population of individuals
    Evaluate each individual from the population
    While (stopping condition)
    {
       Pick the best individuals for reproduction;
       Breed new individuals by means of the crossover;
       Apply a small perturbation over these new individuals;
       Evaluate their individual fitness;
       Replace the worst individuals;
   }
```

Algorithm 2: Basic GA pseudo-code.

3.1.3 Memetic Algorithms

A memetic algorithm (MA) [30] represents one of the current growing areas of research in EC. MAs are a population-based technique for heuristic search in optimisation problems. These are much faster than traditional GAs for many problem domains. Fundamentally, these combine GA's operators with LS heuristics (an LS algorithm typically refines the solution obtained by the GA's operators).

Conversely, the continuous application of LS as a refinement mechanism does not guarantee a better performance. The frequency and the intensity characterise the level of progression (exploration) in opposition to the refinement achieved (exploitation) in the MA search. Thus, a more intense exploitation implies having more chances of convergence to the local optima. Evidently, it highly depends on the stage where the algorithm is, so it is broadly agreed that exploration should be more important at the beginning of the process and exploitation should be performed at the end [31-33]. However, we will see in this work that success can be achieved by dynamically adapting exploration and exploitation, depending on the circumstances found in our search.

For these reasons, some researchers have successfully denoted MAs as Hybrid GAs while others consider them as class of MHs. Frequently, MAs are also referred to in the literature as Baldwinian EAs, Lamarckian EAs, cultural algorithms, or genetic LS.

```
Procedure Basic_Memetic_Algorithm
{
    Generate an initial population of individuals
    Evaluate each individual from the population
    While (stopping condition)
    {
       Pick the best individuals for reproduction;
       Breed new individuals by means of the crossover;
       Apply a small perturbation over these new individuals;
       Evaluate their individual fitness;
       Replace the worst individuals;
       Each g generations, refine the k best individuals;
   }
```

Algorithm 3: Basic MA pseudo-code.

3.2 CLASSIC APPROACHES TO DMAS

Chapter 1 presented the two types of algorithms for DMAS we can find in the state-of-the-art. There exists a kind of ad-hoc algorithms conceived for short-term planning environments in which a permanent planning is required because of the high variability of the system. Instead, there are other techniques devised for more stable (long-term planning) environments. Nevertheless, when facing a dynamic system, these approaches cannot be efficiently applied, since an adaptive method is needed.

In this section, we describe different techniques which could be applied to the problem of workload distribution in dynamic multi-agent systems (stable environments are out of the scope of this dissertation). Note that the purpose of this section is to briefly describe these techniques rather than to deeply detail them as the reader can carefully peruse the references provided in the following subsections, if desired.

3.2.1 Random Workload Balancing

Random workload balancing (RWB) [34] purely assigns a random profile to each agent (among the available ones for that agent). In RWB, the neighbourhood covers the whole search space. After multiple iterations, the best solution found is chosen. Supposing there are *s* possible solutions, the probability of finding the global optimum is *1/s* for each execution. This technique can be appropriate whether there is little communication overhead and numerous agents are available. As the number of agents decreases, the workload of the busiest agents increases in relation to the average agent workload, resulting in poor parallel efficiency. Since each task is assigned to an agent by selecting a random destination, RWB only needs to execute a single pass through the tasks list.

3.2.2 Random Neighbour Search

Differently to LS, random neighbour search (RNS) [35] consists in jumping from a candidate solution to a random neighbour (note that basic LS sequentially explores the neighbourhood). If the hop implies an improvement of the candidate solution, the best solution is updated and then considered as new candidate solution. This process is carried out until a given computing time has been elapsed or a fixed number of random neighbours has been generated.

3.2.3 Greedy Workload Balancing

Greedy workload balancing (GWB) [36] reallocates agents without considering the current assignment *task type-agent* (note that other techniques start out from a neighbour solution but GWB does not). An agent heap is built with the intention that the agent with the least assigned workload is on the top of that heap. In the beginning, no tasks are assigned to any agents, hence every agent in the heap has no workload, and the agent on the top of the heap is randomly chosen. A task heap is also built and organised so that the most time-consuming task is on the top of the heap. For each agent, the most time-consuming unassigned task is allocated to the less loaded agent with the capability to handle that type of task. Afterwards, the agent's workload is updated and both heaps are readjusted. This process is carried out until every task has been assigned to an agent with the required skills.

3.2.4 Skill-Based Routing

Skill-based routing (SBR) [37] is a task-assignment policy to dispense new work items to the most appropriate agent (the appropriateness is problem dependent), rather than to purely select the next available one. Habitually, the routing strategy is led by a simple heuristic (e.g. efficient driven SBR) as SBR claims for quick movements rather than convoluted, time-consuming formulas. SBR usually relies on the Erlang-C formula [38] which has been broadly applied to the CC domain. Nevertheless, some researches [39, 40] claim that the conventional Erlang-C formula is no longer applicable to settling on staff schedules as they are frequently inexact.

3.2.5 Dynamic Programming

Dynamic programming (DP) [41] is a technique which basically breaks problems down into smaller overlapping sub-problems. The philosophy of DP relies on solving problems where we need to find the best decisions serially. DP takes less time than other methods when it is applicable, because the results of certain calculations are stored and can be re-used by succeeding operations.

3.2.6 Branch and Bound

Branch and bound (BB) [42] is a broad-spectrum algorithm devised for discrete and combinatorial optimisation problems. It systematically itemises all candidate solutions, from the uppermost one to the lowest one, discarding unproductive candidates. Every node (candidate solution) at a level *l* in the search tree corresponds to a partial sequence of *p* operations.

3.2.7 Variable Neighbourhood Search

Variable neighbourhood search (VNS) [43] is an MH whose fundamental idea is to cause a systematic, stochastic change of neighbourhood within an LS. VNS escapes from local optima by changing of neighbourhood. To achieve it, VNS increases the size of the neighbourhood until a local optimum, better than the current one, is reached.

3.2.8 Variable Neighbourhood Descent

Variable neighbourhood descent (VND) [44] is an MH where the search is not restricted to only one neighbourhood as in the LS but, instead, it deterministically changes at the same time as the algorithm advances (predefined sizes for the neighbourhoods).

3.2.9 Simulated Annealing

Simulated annealing (SA) [45] is an MH of variable search environment, which generalises Monte Carlo's method. SA proposes that the current state of a thermodynamic system is equivalent to the candidate solution in optimisation, the energy equation for a thermodynamic system is analogous to a target function and the ground state corresponds to the global minimum. This technique has the ability to hinder getting trapped in local optima since the algorithm allows for changes that decrease the values returned by the target function with a given probability. This probability depends on the current temperature value which varies according to the cooling scheme. The main complexity is to determine the right value for the initial temperature and the cooling scheme.

3.2.10 Tabu Search

The meaning of the word *tabu* (also known as *taboo*) refers to a prohibition imposed by social customs as a protective measure [46]. In particular, tabu search (TaS) is based on the principle that search techniques should incorporate adaptive memories and guiding exploration mechanisms. The adaptive memory [47-49] allows for the implementation of procedures that are capable of economically and effectively navigating the search space. These memories introduce complexities that often confound alternative approaches as they allow for restriction of the search environment and the introduction of intensification mechanisms in zones of the search space that have been already visited, or diversification in possible zones of the search space which are rarely visited [49].

3.2.11 Scatter Search

Scatter search (SS) [50] works over a set of solutions (reference points) by merging them in order to produce new feasible ones. The combination of solutions is commonly accomplished in a linear way. These combinations can be devised as a feasible generalisation of the existing solutions.

3.2.12 Iterated Local Search

The basic idea of iterated local search (ILS) [51] is to concentrate the search on a smaller subspace defined by the solutions which are locally optimal to the current one. ILS consists in the iterative application of an LS method. To avoid getting trapped in local optimums, a perturbation is applied before executing each LS.

3.2.13 Multi-Start Search

There are two phases in multi-start search (MSS) [52]: initially, a feasible solution is generated and, afterwards, is normally improved by means of an LS procedure. MSS is relatively simple because it merely executes several LS's from different initial solutions. The stopping condition for each LS is then taken as a restarting criterion. The most imperative disadvantage of improving each solution by means of an LS procedure is the possibility of getting ensnared in a non-optimal local optimum. MSS heuristics are earmarked to obtain limited solutions as far as the LS procedure cannot avoid escaping of non-promising environments. A key issue for the performance of MSS is whether (or not) the information about the topology of the neighbourhood (corresponding to the distance among neighbour solutions) is used.

3.2.14 Greedy Randomised Adaptive Search

The greedy randomised adaptive search (GRASP) [53] is one among those MSS methods whose first phase (constructive phase) randomly generates a greedy solution. The second phase (refinement) iteratively improves every solution by applying an LS procedure. Greedy randomised solutions are generated by injecting new elements to the problem's solution set from a list of elements ranked by a greedy function according to the quality of the solution (problem dependent). This method provides an appropriate and simple framework to develop algorithms for hard optimisation problems. The goal of this methodology is to combine the diversification strategy given by the construction phase with the intensification given in the improvement phase.

3.2.15 Ant Colony Optimisation

Ant colony optimisation (ACO) [54] is a stochastic method which can be applied to problems that can be simplified to finding the right paths within a graph (usually, the shortest ones).

Pheromone is a chemical substance secreted by a living organism that transmits a message inducing other members of the same species to react in a certain way. In our case, virtual ants deposit pheromones once they have built their solutions. The release of such a chemical signal, although systematic, is not constant. It is, instead, dependent upon the heuristic desirability of transition. This pheromone release is carried out once the solution is complete and is only updated when the loop ends. In order to refine the ants´ generated solutions an LS procedure can be added to this algorithm. An ant *a* chooses to go forward to the following node with a determined probability that can be calculated as follows:

$$P_{rs}^{a} = \begin{cases} \dfrac{[\tau_{rs}]^{\alpha} \cdot [\eta_{rs}]^{\beta}}{\sum_{u \in N_r^k} [\tau_{rs}]^{\alpha} \cdot [\eta_{rs}]^{\beta}}, if \ (s \in N_k(r)) \\ 0, other\_case \end{cases} \quad \textbf{(1)}$$

$T_{r,s}$ is the amount of pheromone on edge *r-s*, α is a parameter to control the influence of $\tau_{r,s}$, $\eta_{r,s}$ stands for the desirability of edge *r-s* (classically, $1/d_{r,s}$, where *d* is the distance) and β refers to a parameter which controls the influence of $\eta_{r,s}$.

3.2.16 Particle Swarm Optimisation

Particle swarm optimization (PSO) [55] is a technique which does not require any knowledge of the gradient of the problem to optimise. PSO emulates the behaviour of a group of birds which are flocking. PSO keeps a population of candidate solutions (particles) and then shifts them around in the search space in accordance with a more or less straightforward formula.

3.3 DISCUSSION

Section 3.2 has presented different techniques which could be somehow applied to a DMAS based on blackboard architecture.

We can distinguish methods based on LS from those rooted in global search (GS). GS takes into account the whole search space whereas LS approaches can be applied to problems which can be devised as finding a solution maximising (or minimising) a criterion among a number of candidate solutions. An LS algorithm starts out from a candidate solution and, thus, iteratively moves to a neighbour solution, generating the neighbourhood until a solution deemed optimal is reached or a predefined amount of time has been elapsed. The main problem with LS methods is that these usually get stuck in local optimums which are often far from the global optimum. This setback can be mainly mitigated in five distinct ways:

1) The first possible solution, exemplified by VND, is to modify the environment (also known as neighbourhood). In VND, the search is not only restricted to one environment as LS imposes; instead, the size of the environment deterministically changes as the algorithm progresses. The change of environment is a technique that is dependent upon the stage at which the algorithm is currently working.

2) The second possible solution is to permit deterioration movements, such as in SA or TaS. In the SA method, each point of the search space is equivalent to a state of some physical systems, and the function *E(s)* to be minimised is similar to the internal energy of the system in that state. The aim is to bring the system, from a random initial state, to a state with the smallest amount of energy. TaS increases the performance of an LS method by employing memory structures. Once a potential solution has been reached, it is marked as *tabu* so that, the algorithm does not visit that possibility recurrently.

3) The third possible solution is to restart from another initial solution as MSS, GRASP, ILS or VNS do. In the case of the MSS, initial solutions are randomly generated and, afterwards, the algorithm applies an LS over them as a fine-tuning mechanism. This is equivalent to executing several LS in parallel. Therefore, the accuracy of the results will depend upon the number of executions that are launched. However, this is an inefficient method because a conscious stopping condition has to be provided. Conversely, ILS

applies a mutation operator before each execution to attain an intermediate solution which is refined by an LS. VNS (very similar to VND) is an ILS method which changes of environment when the solution obtained is worse than the current one. Finally, GRASP relies on the use of a randomised greedy in its basic version.

4) Another way to find a good solution involves using methods based on populations, such as GAs and MAs. If the diversity of the population is low, then the GA converges to the closest neighbour. In contrast, if the selective pressure is high, which makes the diversity low, individuals will be alike or even identical. GAs are a powerful GS technique that slowly converges to the global optimum for a set of relevant real-world problems. MAs emerge as an improvement of this mechanism in which an LS is applied over a subset of individuals each n-generations.

5) Finally, there are other strategies to obtain a feasible solution such as constructive methods (e.g. ACO).

The MHs presented above provide diverse methods to escape from local optima. The empirical impact of these MHs has been immense. Diverse tendencies on MH schemes have been explored by many researchers. The most relevant issue, provided by the incorporation of such techniques, is to know whether the benefit of the performance enhancement compensates for the effort of its implementation.

Frequently, trendy appealing heuristics are skilfully figured out. Also, great effort and inventiveness has been deployed in the adjustment of numerous parameters, but as yet the reasons that make them work still remain unknown. When facing a dynamic real-world production environment, some techniques (we will present an empirical study in Chapter 5) cannot perform well-enough.

Intuitively, although RWB and RNS require low computing times, will not be appropriate for a real production environment as they do not guarantee an accurate solution and are not robust enough. A randomly generated solution can be acceptable as an initial solution, but not as a proper search mechanism. With luck on our side, we might find a good solution, but we would rapidly notice that these methods do not always perform properly. In fact, the probability of obtaining the global optimum is *1/nsl* where *nsl* stands for the number of possible solutions in the search space. Imagine a die with *nsl* faces (a very large number) with the added problem that we can only throw that die *nt* times in each time frame (where *nsl* >> *nt*).

GWB and SBR “route” work items to available agents by applying basic heuristics, considering the current state of the system. Obviously, these techniques can be perfectly employed in DMAS but these fast, unplanned decisions may guide the algorithm to congestion states (evident need of a better planning which takes into account future states).

In basic LS, a neighbour is generated every iteration. Theoretically, due to its local character, it is difficult to reach a high-quality solution because, when a local minimum is found, the algorithm will often stagnate as deterioration movements are not allowed.

In VND and VNS, the search is not simply restricted to a single search environment, but instead, the environment changes as the algorithm advances (deterministically in VND and stochastically in VNS). Therefore, the improvement of basic LS is remarkable as we will verify in Chapter 5.

TaS and SA introduce a very sophisticated mechanism of deterioration movements. However, these techniques only better perform when the time frame is not too reduced: SA takes time even when we apply Cauchy’s scheme which is the fastest one and TaS requires of many iterations to take advantage of using the memory structures.

MSS increases the probability of finding an accurate solution compared to basic LS as many LS’s are run in parallel. In contrast, GRASP improves this philosophy by means of a probabilistic greedy procedure. This greedy process reassures us that, on one hand, that initial solution will be more or less promising and, on the other hand, that other local minima may be found, since the algorithm can start from different initial solutions.

Constructive techniques (e.g. ACO), although they are a very promising growing area, are not fast enough to be applied to real-world DMAS as [56] demonstrates.

Finally, GAs offer a different mechanism to finding precise solutions based on a population schema. Generally, GAs converge very slowly to the global optimum (or optima) but, when these are combined with LS procedures (MAs), GAs are an astonishingly powerful search technique. Our approach relies on MAs as we will see in Chapter 4.

Note that we have presented all techniques from a theoretical point of view, but we will also compare most of these techniques in Chapter 6 (at least, one representative from each policy) to determine the appropriateness of each strategy for a real-world DMAS. To conclude, Table 2 shows the efficacy of each strategy to find a nearly optimal solution for a real-world DMAS in relation to the time-frame considered.

Table 2: Comparison of strategies' efficacy for a real-world DMAS in relation to the time-frame considered. MA is highlighted because it is our choice for DMAS.

Algorithm	*Efficacy short-term*	*Efficacy middle-term*	*Efficacy long-term*
RWB	low	low	Low
RNS	low	low	Low
Basic LS	low	medium	Low
GWB	medium	low	Low
SBR	medium	low	Low
DP	low	low	high
BB	low	low	high
VNS	low	medium	medium
VND	low	medium	medium
SA	low	medium	high
TaS	low	medium	high
SS	low	medium	high
ILS	low	medium	medium
MSS	low	medium	medium
GRASP	low	medium	medium
ACO	low	low	medium
PSO	low	low	medium
GA	low	medium	medium
MA	low	high	high

CHAPTER 4. A NOVEL APPROACH TO THE PROBLEM OF WORKLOAD DISTRIBUTION IN DMAS

This chapter proposes a new approach to the problem described in Chapter 2. Section 4.1 sets out the bases of this novel approach. Section 4.2 describes the methodology that we have followed. Section 4.3 focuses on the forecast component which supports the first module of our approach. Section 4.4 addresses the second component of our approach; in other words, the search module. In Section 4.4, we also propose multiple mechanisms to maintaining a fair balance between diversity and intensity in simple and parallel genetic algorithms when optimising.

4.1 UNDERLYING IDEA

We have illustrated in Section 1.1 how short-term planning techniques distribute arriving tasks to the existing available agents having the required skills to process them by employing greedy heuristics while long-term planning techniques contend with a stable incoming task flow and a longer stability over time which is not the archetypal situation in nowadays DMAS's. In this way, short-term planning strategies distribute the workload without considering future system states (just the current system configuration), provoking inapt allocations *task-agent* for near future. In contrast, long-term planning strategies find optimal solutions for a given system state. However, if the system is not very stable, we might have serious problems in the future, because an optimal configuration for the current system state may not be the best option in the future as these algorithms take time to reach a solution. In other words, we might be using an optimal system configuration for a completely different system state.

This section puts forward the bases of our approach to DMAS. The underlying hypothesis of this work, which will be demonstrated and confirmed in Chapter 6, is that DMAS’s require precise (nearly optimal) allocations of task types to the right available agents for each (*adaptive*) middle-term time-frame over time rather than continuous naive/greedy assignments for each system state or static long-term configurations for remote future system states (see Figure 1).

We can now remain generic and develop a more stylish approach by reformulating the standard variant of the problem of workload distribution in DMAS as the concept of adaptive time-frame has been already introduced. We basically need to enlarge (or reduce) the observed time-frame and then forecast the real system state in a future point in order to apply more sophisticated search algorithms which can outperform both short-term and long-term planning strategies. There, a need of an exact prediction of a middle-term system state comes out. Subsequently, a search algorithm must find a feasible solution for the predicted system state by reaching a fair balance between diversity (exploration) and intensity (exploitation) in order to meet with success.

Before explaining each “box” of our approach, we will present the overall process in order to clarify the steps we take. We firstly need to analyse the dynamism of the system within a given period of time with the purpose of determining the right time-frame size. This time-frame cannot be fixed and its size must change over time if there are changes in the behaviour of the system (variability in the arriving task flow). Once we have determined the size of the time-frame, we need to forecast the state of all variables at the end of the time-frame (if we are at time t, we will forecast all system variables at time $t+v$). Given our predictions, we need to optimise the allocation of existing tasks to the available agents having the required skills (see Figure 5).

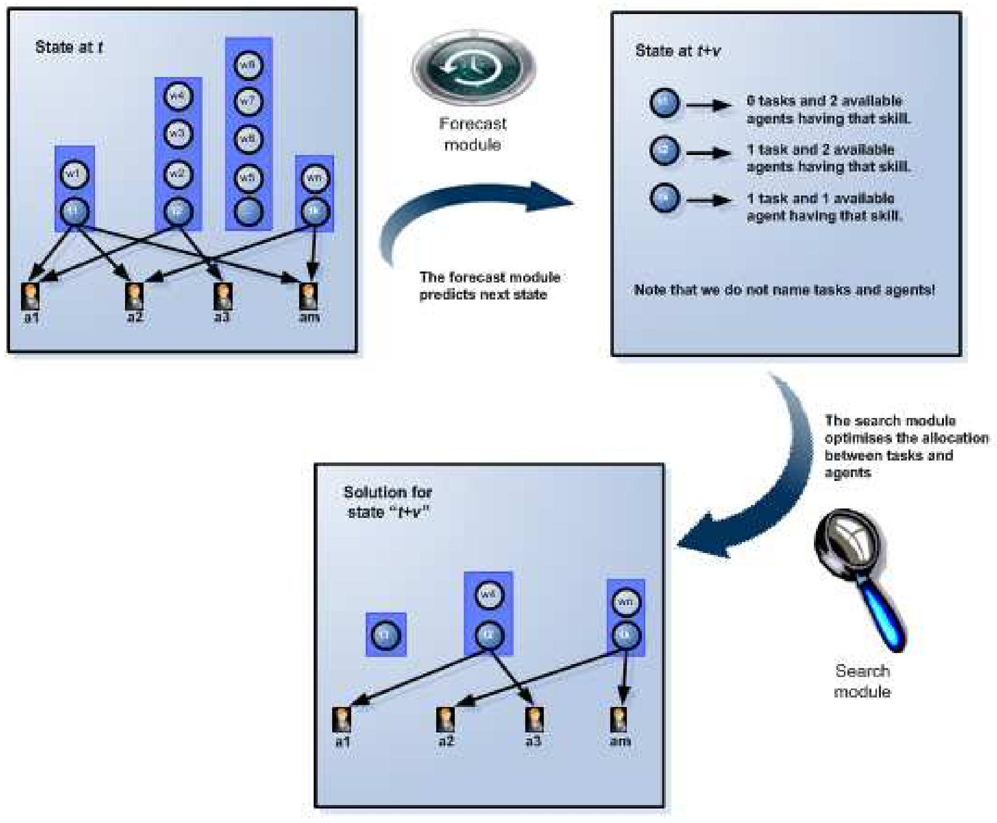

State at *t*
w1
w2
w3
w4
w5
w6
w7
w8
wn
t1
t2
tk
a1
a2
a3
am
Forecast module
The forecast module predicts next state
State at *t+v*
0 tasks and 2 available agents having that skill.
1 task and 2 available agents having that skill.
1 task and 1 available agent having that skill.
Note that we do not name tasks and agents!
The search module optimises the allocation between tasks and agents
Search module
Solution for state "*t+v*"

Figure 5: Overall process → forecast module + search module.

The Algorithm X also describes the steps of our approach to better explain the underlying idea.

```
Procedure "Approach to the problem of workload distribution in DMAS"
{
   Analyse the dynamism of the system;
   Set up a size for the time-frame;
   Forecast all variables for next state; //num. of tasks (by type), agents available, etc.
   Optimise the assignment among predicted tasks (by type) and predicted agents;
   Go to next state;
}
```

Algorithm X: Overall procedure.

The two most complicated factors to develop our approach are: the determination of the size of the time-frame (we continuously analyse the system dynamism to determine the right size for the time-frame) and the state-transition function (when system dynamism changes).

To address the first difficulty (size of the time-frame), we can set up predefined time-frame sizes (see Figure 6), depending on the dynamism level of the system (e.g. *3* levels: *low*, *medium* and *high*). Obviously, this choice must be done according to a previous, exhaustive statistical analysis. In our experiments, we have considered *5* levels depending on system variability: *very low* (*v=3000* seconds), *low* (*v=1500* seconds), *medium* (*v*=300 seconds), *high* (*v*=120 seconds) and *very high* (*v=60* seconds). The point of selecting predefined time-frame sizes is given by the requirement of robustness that real-world DMAS's habitually impose. If we enable the system to automatically assign any size for the time-frame, we may crash down the system (we may have *2*-second time-frames or *20*-hour time-frames which might seize up the system).

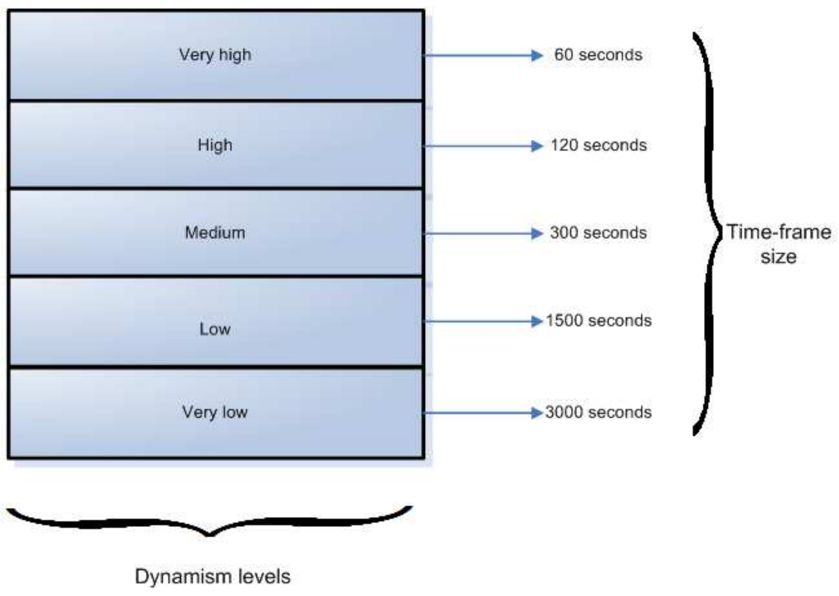


Figure 6: Time-frame sizes depending on the dynamism level.

To deal with the second complexity (when to change of dynamism level), we must determine the right state-transition function. This is a problem-dependent task and we cannot claim any universal rule of thumb. Instead, we propose some guidelines to accomplish with this arduous task. During the statistical analysis, we encourage the reader to analyse smaller intervals than the time-frame (let's say *30* seconds). Then, we should break down this interval into subintervals (e.g. *5* subintervals of *6* seconds) and plot a time series. If a given point highly differs from the previous one, we should not activate the state-transition function as peaks may crash down the system. But, when the trend of the time series drawn by these consecutive points shows important oscillations, we should switch to another dynamism level (e.g. if the tendency shows an important dynamism decrement, we should then enlarge the time-frame by switching to a less dynamic level). In summary, we take into account the trend of these consecutive points as well as the dispersion among them. Figure 7 shows a *300*-second time-frame. For this time-frame, we analyse a smaller (shifting) time-frame of *30* seconds with *5* equidistant consecutive points (each *6* seconds). If we plot these points, we would have a time series (number of arriving tasks at each time point). We may discover numerous possible situations but, in this figure, we have only illustrated *6* different cases (bear in mind that a rigorous statistical study must be performed to achieve it). Figure 7.A shows a very changing time series (high dispersion without well-defined trend); therefore, the dynamism level would be *very high*. Figure 7.B exemplifies a quite dynamic time series but with fewer changes than Figure 6.A, so the dynamism level would be *high*. Figure 7.C and Figure 7.D point up *medium* dynamism level because there is a clear trend (increasing in Figure 7.C and decreasing 7.D). Figure 7.E illustrates a time series without changes, therefore the dynamism level should be *very-low*. Finally, Figure 7.F presents a time-series with few changes, thus, the dynamism level would be *low*.

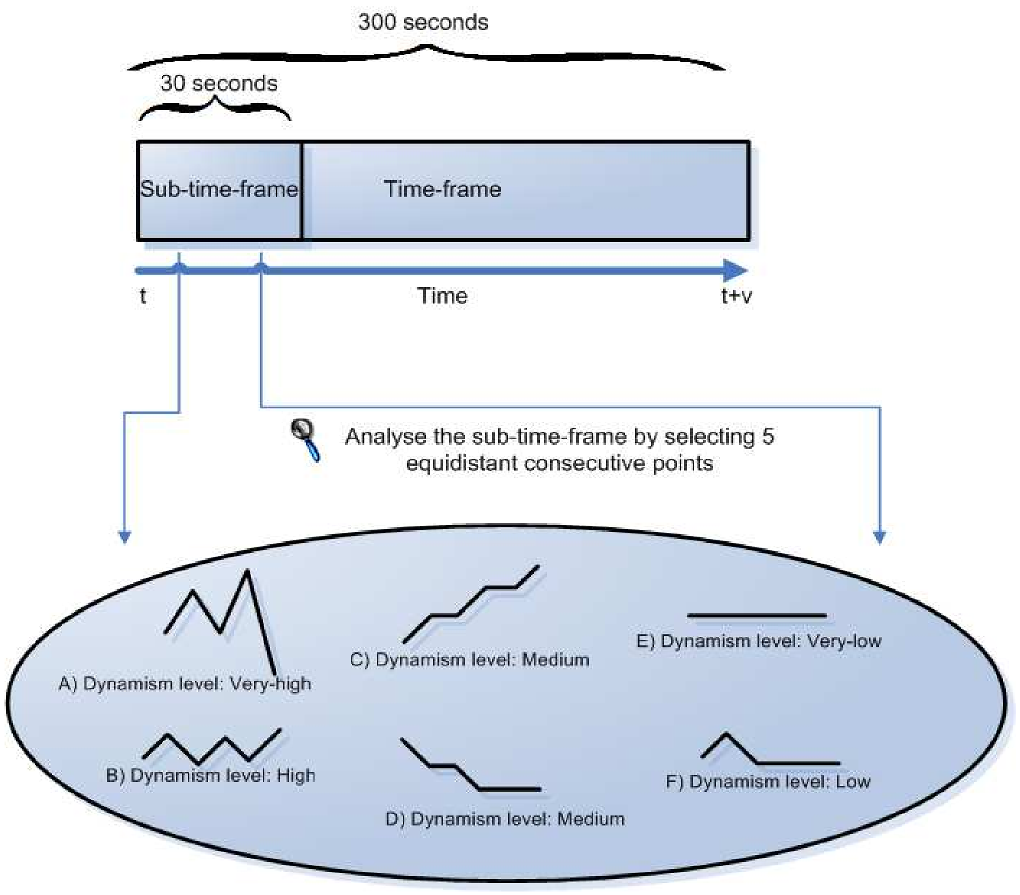

300 seconds
30 seconds
Sub-time-frame
Time-frame
t
Time
t+v
Analyse the sub-time-frame by selecting 5 equidistant consecutive points
A) Dynamism level: Very-high
B) Dynamism level: High
C) Dynamism level: Medium
D) Dynamism level: Medium
E) Dynamism level: Very-low
F) Dynamism level: Low

Figure 7: Analysing the time-series within a shifting sub-time-frame.

Naturally, the smaller window must be shifted according to the time (never employ fixed windows). Finally, we do not impose "sequentiallity" when changing of dynamism level as Figure 8 exemplifies. Arrows symbolise that we can reach every state from any other state. Circles represent the dynamism levels. Note that we have not plotted self-pointing arrows as no transition is needed.

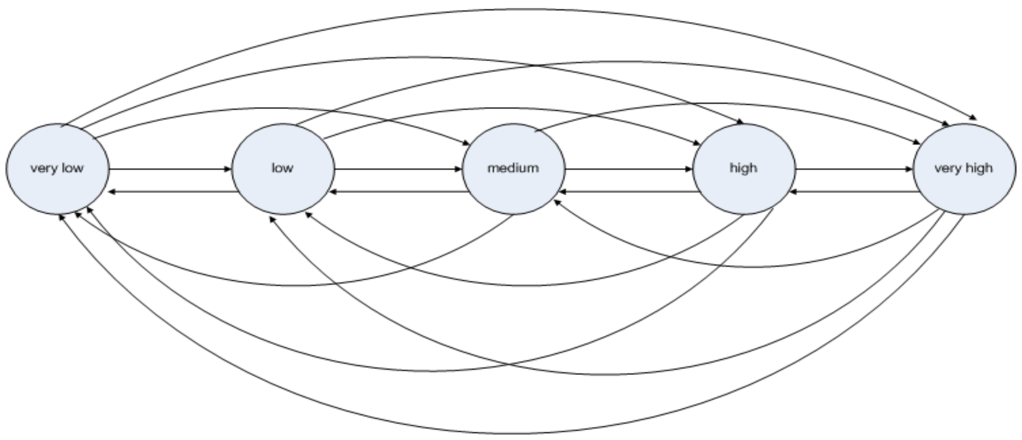


Figure 8: Potential dynamism level transitions.

Now, we have the mechanism to determine the size of the time-frame as well as method to track the dynamism level. From this information, we need to forecast all system variables for time *t+v*. Obviously, even when the forecast is pretty accurate, we are introducing some noise to the model as we are searching a nearly optimal solution for a predicted system state which may slightly differ from the real future state. In order to mitigate the impact of that noise, it is crucial to design a powerful forecast module which can provide us with the best possible approximation of next future system state. From these accurate predictions, we will apply a search algorithm based on a parallel memetic algorithm to discern fine allocations *task type-agent* from inappropriate ones.

Until now, we have presented copious numbers of variables from an ordinary DMAS in Section 2.2. Although we have to consider all these variables in order to attain a feasible solution, uncertainty chiefly comes from the number of pending tasks grouped by task types and the number of existing available agents having each skill. As tasks continuously appear and require of a certain processing time to be executed, and given that the size of the time-frame *v* is variable (a smaller number when there is great dynamism or a larger number when there are few oscillations); we can assume that the number of pending tasks and available agents at time *t+v* depends upon the number of pending tasks and available agents at time *t* as some tasks may not be accomplished during these *v* seconds. Actually, we will notice that a system state at time *t+v* only depends on the system state at time *t* as it follows a Markov process [57].

Now, let's formalise the definitions exposed along this section. Denote the initial state at time *0* as ξ_0 where we know all system variables (W_0, T_0, A_0, S_0, P_0). We just mean that at the beginning (time *0*), we know the number of pending tasks, their types, the number of available agents, the potential skills and the prospective profiles (we highly encourage the reader to briefly review Section 2.2 to refresh the meaning of each variable). Also, denote the current state at time *t* as ξ_t (W_t, T_t, A_t, S_t, P_t) and designate next future state at time *t+v* as ξ_{t+v} ($W_{t+v}, T_{t+v}, A_{t+v}, S_{t+v}, P_{t+v}$).

Finally, denote the state-transition function as $\delta: \xi_t / v_{dynamism\_level} \rightarrow \xi_{t+v}$. This just means that every state ξ_{t+v}, depends on the previous state ξ_t and the transition occurs each v seconds (size of the time-frame which depends on the dynamism level). The variables of a given state are directly visible to the observer.

Note that short-term planning strategies consider intermediate systems states ($\xi_t < \xi_{intermediate} < \xi_{t+v}$), whereas long-term planning strategies take into account posterior system states ($\xi_t < \xi_{t+v} < \xi_{posterior}$). Therefore, short-term planning strategies rely on smaller transition-state steps δ than our approach, while long-term planning strategies hinge on longer transition-state steps δ than our approach. Graphically, this characteristic can be seen in Figure 9. Notice that ξ_i stands for $\xi_{intermediate}$ and ξ_p represents $\xi_{posterior}$. Also, note that we have not plotted ξ_0. In a real-world production environment, we can set up an initial system configuration for ξ_0 that considers historical records. From this first configuration for the initial state ξ_0, we should employ the mechanism we propose (prediction window determination based on system dynamism + current system state prediction + optimisation).

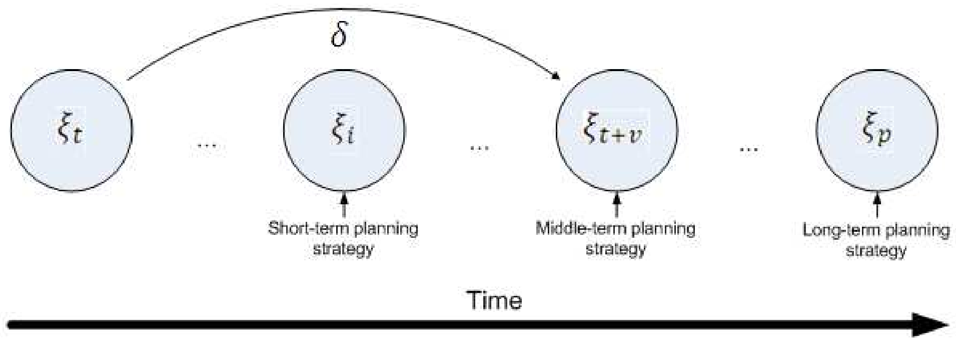

δ
ξ_t
...
ξ_i
...
ξ_{t+v}
...
ξ_p
Short-term planning strategy
Middle-term planning strategy
Long-term planning strategy
Time

Figure 9: System states depending on the time-frame considered.

The key purpose of the rest of the present chapter is to provide a solution for the problem of workload distribution in DMAS, given the reformulation of the problem that we have proposed in this section. Our approach combines predictions (for middle-term system states of DMAS's by means of an upgraded resilient back-propagation neural network) with a powerful search mechanism (founded on a parallel memetic algorithm).

4.2 METHODOLOGY

This section describes the methodology employed by our approach. The first step consists in determining the size of the time-frame based on system variability (*very low*, *low*, *medium*, *high* and *very high*) as explained in Section 4.1. Once the right size of the time-frame has been detected, we must forecast all variables of next system state at time $t+v$, ξ_{t+v}. These predictions are made by means of a forecast module which relies on an upgraded resilient back-propagation neural network.

Given the predictions from the forecast module, the search module, implemented as a parallel steady-state MA, optimises the assignment among task types and agents. We propose an island topology and migration operators for individuals exchanging. We will consider a master island and several slave islands. Each island corresponds to a single MA. Each MA maintains a set (population) of abstract representations (chromosomes) of candidate solutions (phenotypes) to the problem described in Chapter 2. The population is partially randomly initialised (see Section 4.2.4 to obtain further information). Then, its individuals are evaluated by applying a fitness function over them. From this population, some individuals are selected and, then, recombined (crossover). Subsequently, the offspring may suffer mutations in some genes. Afterwards, some of these individuals replace others from the population according to the replacement scheme. Every generation includes all previous actions. Finally, an LS mechanism is applied over a percentage of the population each g generations. All these steps are carried out until a predefined time has been elapsed.

4.3 FORECAST MODULE

This section describes how to model a forecast module with an upgraded resilient back-propagation neural network to predict all unknown variables at time $t+v$, where t is the current instant and v is the size of the time window. Section 4.3.1 surveys the state-of-the-art on DMAS forecasting. Section 4.3.2 provides the required background to understand our learning algorithm for neural networks. In Section 4.3.3, we formulate the mathematical bases to define learning algorithm for neural networks. Section 4.3.4 explains how to fine-tune neural networks to DMAS forecasting, given our innovative learning algorithm.

4.3.1 State-of-the-art on Forecasting

Most people perceive the world as a place where there are a large number of alternatives. In this context, forecasting refers to the estimation of output values in unknown situations to help decision making and planning. But, what does forecasting stand for in a DMAS domain? Forecasting refers to the estimation of values at certain specific future times. In this manner, there are many things that would be desirable to predict in a common DMAS such as arriving tasks, task failures, available agents having a certain skill, working levels (this is the time the agent is truly processing tasks), service rates (given by a quality metric which depends on the domain) and average delay times.

Why is it interesting or necessary in a DMAS domain? Particularly, a precise prediction enables us to be prepared for the future to correctly balance workload among agents, presenting higher service levels and, eventually, optimising our resources. We can compile arriving tasks, task failures and queuing tasks in a unique value, the number of tasks (grouped by task types) to handle.

Unfortunately, there is no way to state what future will bring along with complete sureness. *Risk* (wrong predictions generally entail losses of money or even major hazards) and *uncertainty* (ambiguity or indecision to accomplish our predefined goals) are omnipresent in forecasting to the degree that it is customarily considered good practice to specify the level of uncertainty linked to forecasts.

A significant but ignored facet of forecasting is the close liaison it holds with planning. Forecasting can be expressed as predicting what future will resemble, whereas planning enlightens what future should look like. There is no universal, suitable forecasting method to use, as it depends on our objectives and preconditions.

There are a wide variety of forecast techniques for DMAS's, although we will focus on the most relevant ones along the present section. In this section, we want to introduce and thus discuss these relevant forecast techniques, presenting their positive and negative characteristics.

This section is organised as follows: Subsection 4.3.1.1 addresses the classical Poisson distribution. Subsection 4.3.1.2 briefly exposes regression techniques. Subsection 4.3.1.3 presents some time series methods. Subsection 4.3.1.4 covers this problem from neural networks' point of view.

4.3.1.1 *Poisson Distribution*

Traditionally, incoming call forecasting in CCs has been approximated according to a Poisson distribution (PD). Nevertheless, PD can be perfectly applied to other DMAS's which accomplish several assumptions that we are going to expose in the present section. PD expresses the probability of a number of events occurring within a time-interval, when these are independent of the previous event and occur with a known rate. Under these conditions, it is a reasonable approximation of the exact binomial distribution of events. Additionally, PD provides a useful mechanism to assessing the percentage of time when a given range of results are expected. In the calculation of the distribution function, the values for the mean and standard deviation are carried over from the binomial distribution.

Assuming pure-chance arrivals and pure-chance terminations leads to the following probability distribution:

$$P(n) = \left(\frac{\mu^{n}}{n!}\right)e^{-\mu} \tag{2}$$

where *n* denotes the number of arriving tasks in an interval of duration *d*, *μ* stands for the mean of arriving tasks at time *t* and *e* refers to the base of the natural logarithm (e ≅ 2.7183). Thus, "conventional" approaches assume that the number of arriving tasks at a given time, *t*, follows a PD. For this reason, pure-chance traffic is also named as Poisson traffic.

Table 3 shows the values returned by a PD when varying λ between *0.1* and *1.5*.

Table 3: Poisson distribution when varying λ between *0.1* and *1.5*.

n\λ	*0.1*	*0.5*	*1*	*1.5*
0	0.905	0.607	0.368	0.223
1	0.090	0.303	0.368	0.335
2	0.005	0.076	0.184	0.251
3	0.000	0.013	0.061	0.126
4	0.000	0.002	0.015	0.047
5	0.000	0.000	0.003	0.014
6	0.000	0.000	0.001	0.004
7	0.000	0.000	0.000	0.001
8	0.000	0.000	0.000	0.000
9	0.000	0.000	0.000	0.000
10	0.000	0.000	0.000	0.000

In the same way, Figure 10 plots the points of Table 3 to better understand PD's nature.

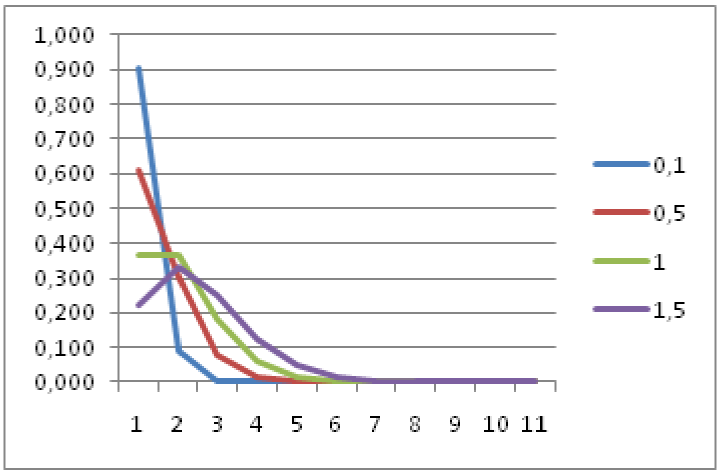


Figure 10: Poisson distribution when varying λ.

PD has inspired other authors to extend its idiosyncrasy to other distributions. Reviewing the literature, we can bump into numerous algorithms founded on (or merely supported somehow by) a PD. As an example, we should highlight Erlang-based algorithms.

The Erlang distribution, first pioneered by A. K. Erlang [38], is a continuous probability distribution with extensive applicability. This distribution, which has a positive value for all real numbers greater than zero, is given by two factors: the shape *k* (a non-negative integer) and the rate λ (a non-negative real number). The distribution is sometimes defined using the inverse of λ, the scale μ. This distribution appeared as a mechanism to inspect the number of arriving tasks which might simultaneously arrive to the agents of a DMAS. This work, which was originally conceived for the CC domain, has been afterwards extended to other queuing environments by other authors [58, 59].

Figure 11 plots Erlang distribution for $k = 2, 3$ and $\lambda = 3, 1$.

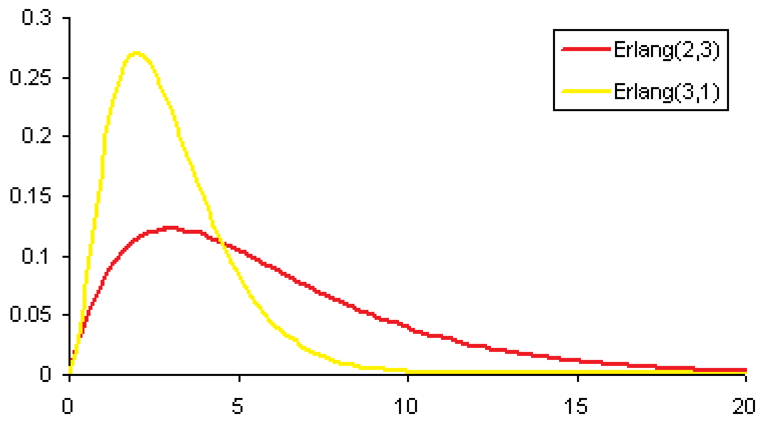


Figure 11: Erlang distribution for $k = 2, 3$ and $\lambda = 3, 1$.

In an Erlang distribution, events are modelled in accordance with a Poisson process and independently occur with some average rate. The waiting times between *k* occurrences of the event are Erlang distributed.

However, the prediction of arriving tasks in a DMAS does not often adjust to a PD with deterministic rate. In all studies (e.g. [60]), the arrival process agrees with a Poisson process only if the arrival rate of the Poisson process is itself a stochastic process.

Characteristically, the variance of arriving tasks within a given interval is much larger than the mean. However, PD's hypothesis states it should be equal to the mean for PDs. The mean arrival rate also strongly depends on the day-time and often on the week-day, but Poisson processes comply with the memoryless property of the exponential distribution [60], which is unable to detect this kind of features. Besides, in some DMAS such as MSCCs, there is positive stochastic dependence between arrival rates in successive periods within a day and arrival volumes during successive days. Taking into account all these premises, we can realise how pertinent is to find a more effective method to forecast which does not just rely on the hypothesis of a simple PD.

Considering these premises, we can become conscious of the need of finding a more effective method to forecast.

4.3.1.2 Regression Model

A regression model (RM) [61] is a statistical method in which an unknown variable is predicted according to its relation with the rest of well-known variables (also named as predictors), using a formula called *regression equation*. This equation deals with some constant parameters which must be optimised to reduce the mean square error (MSE) between the predicted output and its real value. In particular, we study lineal regression (LR) which is one of the commonest variants (actually, we will additionally examine neural networks which can be faultlessly included in multiple regression). LR fits all parameters by applying diverse policies. The commonest policies are the following ones: least squares approach, minimisation of the "lack of fit" and minimisation of least squares loss function as ridge regression assumes. Least squares and linear model are intimately related although these are not identical.

LR approximates the unknown variable with a straight line by using well-known variables as follows:

$$Y_i = \beta_0 + \sum \beta_p X_{ip} + \varepsilon_i \tag{3}$$

where parameter *i* is the pattern-position in the dataset, *p* indicates the *n*-th well-known variable, β_P represents the associated parameters to the *n*-th well-known variable, β_0 is a constant parameter, *Y* refers to a dependent variable and ε denotes the associated error. β_P and β_0 are calculated in order to reduce $\Sigma\varepsilon_i$, using predefined patterns.

The main advantage of this method is the clearness to understand and track the model. Nevertheless, it is hard to choose the variables to generate the model, considering seasonality and trend, which is crucial to better understand the behaviour of a DMAS.

4.3.1.3 Time Series

A time series (TS) is a sequence of observed variables, taken in regular time-slices. This sequence is used for understanding and forecasting the behaviour of a given variable over time based on previous states [62]. A TS approximates future values by applying a (more or less complex) regression to the *n*-previous variables to estimate forthcoming values. TS can be divided into two major groups: exponential smoothing (ES) and autoregressive integrated moving average (ARIMA). At the same time, ES methods, which assign decreasing weights to each previous observation, are divided into: simple time series (SES), dumped trend time series (DTTS) and stationary time series (STS).

SES, or Single Exponential Smoothing [63], is a method for forecasting whether the mean is stationary or slowly changes over time. The name is frankly ambiguous, given that this is a moving average method in which weights decline as the interval between the current time increases. The smoothed value lags the current value as far as this method depends on previous values. When the smoothing value is small, the oscillations are seriously damped and the smoothed value tends in the direction of the mean. Nevertheless, when the smoothing value is large, the oscillations noticeably fluctuate and, as a result, the smoothed value tends to the current value. SES can be obtained as follows:

$$S_t = \alpha y_{t-1} + (1-\alpha)S_{t-1}, \quad 0 < \alpha \le 1, t \ge 3; \qquad F_{t+1} = \alpha y_t + (1-\alpha)S_t, \tag{4}$$

where *S* stands for the smoothed observation, *t* refers to an index which denotes a time-period, α is a constant which must be estimated with the purpose of minimising the MSE and *y* is the observation.

Instead, DTTS or Holt's linear model (also known as *double exponential smoothing*) [64] extends exponential smoothing by incorporating a term for linear trends. This technique is also called "double exponential smoothing". Suppose that at time *t*, y_t is observed, the level L_t is estimated and the slope b_t is known in the series. Afterwards, a *k*-step ahead forecast is $F_{t+k} = L_t + b_{tk}$. DTTS allows us to adjust the slope with each new observation. DTTS can be formalised as:

$$\begin{aligned} &S_t = \alpha y_{t-1} + (1-\alpha)(S_{t-1} + b_{t-1}), \quad 0 \le \alpha \le 1 \\ &b_t = \gamma(S_t - S_{t-1}) + (1-\alpha)b_{t-1}, \quad 0 \le \gamma \le 1; \qquad F_{t+m} = S_t + mb_t \end{aligned} \tag{5}$$

STS, or Holt-Winters' Trend and Seasonality Model [65], is a suitable technique to deploy when data show trend and seasonality. This technique introduces a third equation to cope with seasonality. STS can be formulated as follows:

$$\begin{aligned} &S_t = \alpha \frac{y_t}{I_{t-L}} + (1-\alpha)(S_{t-1} + b_{t-1}), \quad 0 \le \alpha \le 1; \\ &b_t = \gamma(S_t - S_{t-1}) + (1-\gamma)b_{t-1}, \quad 0 \le \gamma \le 1 \\ &I_t = \beta \frac{y_t}{S_t} + (1-\beta)I_{t-L}; \qquad F_{t+m} = (S_t + mb_t)I_{t-L+m} \end{aligned} \tag{6}$$

where *y* is the observation, *S* stands for the smoothed observation, *b* is the trend factor, *I* indicates the seasonal index, *F* denotes the forecast at *m* periods ahead, *t* refers to an index which denotes a time period and *α*, *β* and *γ* are constants which must be estimated with the purpose of minimising the MSE.

The main advantage of Exponential Smoothing TS is that it requires short computing times [66]. Nevertheless, the model cannot accurately predict for a long timeslice [65]. To mitigate this handicap, we generate a daily model to forecast the following day. Another setback of this technique is its low performance when there is a trend as the single coefficient alpha is not enough to fit the prediction.

Differently, ARIMA [65] is determined by three parameters *(p, d, q)*, where *p* is the autoregressive term, *d* is the number of previous values and *q* is the average moving parameter. ARIMA (p, d, q) can be calculated for a TS sequence Y_t *(t=1,2,…, n)*, as follows:

$$\begin{aligned}
&\phi(B)(1-B)^d Y_t = \theta(B) Z_t;\\
&\textit{where } ,\phi(B) = (1-\alpha_1 B - \alpha_2 B^2 - \ldots - \alpha_p B^p)\\
&\textit{and } \theta(B) = (1-\beta_1 B - \beta_2 B^2 - \ldots - \beta_q B^q)\\
&\textit{and } \mathrm{Z}_t \text{ is a white noise sequence and B is the backshift operator}
\end{aligned} \tag{7}$$

ARIMA (p,q,d)·(P,D,Q) represents a multiplication of two ARIMAs to inject seasonality to the model. This method requires that new seasonal and non-seasonal parameters will be estimated; analogously to simple ARIMA. The involved parameters are the following ones: *p* is the autoregressive order which indicates the number of parameters of *φ*, *d* is the number of times that data series must be distinguished to induce a stationary series, *q* is the moving average order which designates the number of parameters of *θ*, *P* is the seasonal autoregressive order that specifies the amount of parameters of *φ*, *D* is the seasonal moving average order which points out the quantity of parameters of *θ*, and *Q* is the number of times that a data series needs to be differenced to induce a seasonal stationary series.

The principal advantage of ARIMA TS is that it usually suites better than Exponential Smoothing TS, although this model requires long computing times [68] and poorly forecast for large time-horizons [69]. To mitigate this handicap, we generate daily models to forecast the forthcoming day as explained in Chapter 6.

Considering these premises, we can realise how promising to forecast data with no trend or seasonal patterns exponential smoothing is. Instead, Holt's method should be applied whether there is a linear trend. For shifting data, exponential smoothing is remarkably well-adjustable, although its speediness depends upon *α*.

4.3.1.4 Artificial Neural Networks

An artificial neural network (ANN) is a mathematical model founded on the operation of biological neural networks [70]. In this manner, an artificial neuron is a computational model inspired in biological neurons and also the simplest processing element of an ANN. Natural neurons receive signals through synapses placed on the dendrites. When the arriving signals surpass a certain threshold, the neuron is activated and emits another signal through the axon. This signal can be sent to another synapse and then activate other neurons.

In order to emulate biological neurons, the artificial ones (see Figure 12) are organised into two units: the first one is a *nonlinear weighted sum* of weight coefficients and input signals, *F(x)*, whilst the second one follows a nonlinear function, widely known as *neuron activation function*, *K*. The function *F*(X) accumulates weights w_i and maps results to an output as given below:

$$F(x) = K \times \left(\sum_{i=1}^{n} w_i \times input_i \right) \qquad \textbf{(8)}$$

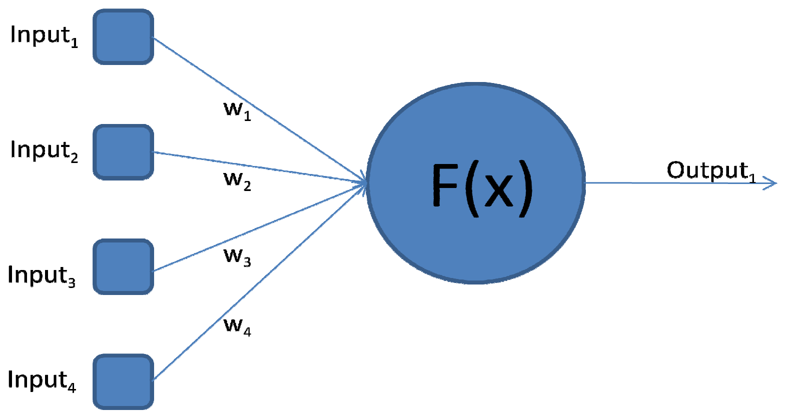


Figure 12: Basic artificial neuron

The weights, w_i, are randomly initialised and then updated during the training process.

There are numerous functions to approximate *K*, but the most widespread ones include the Gaussian function, the hyperbolic function and the sigmoid function. We will employ the sigmoid function as this is the most appropriate one for our dynamic environment. The sigmoid function and its derivative are defined as indicated below:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{d\sigma(x)}{dx} = \sigma(x) \times (1 - \sigma(x)) \qquad \textbf{(9)}$$

Figure 13 plots the sigmoid function to facilitate reader's understanding.

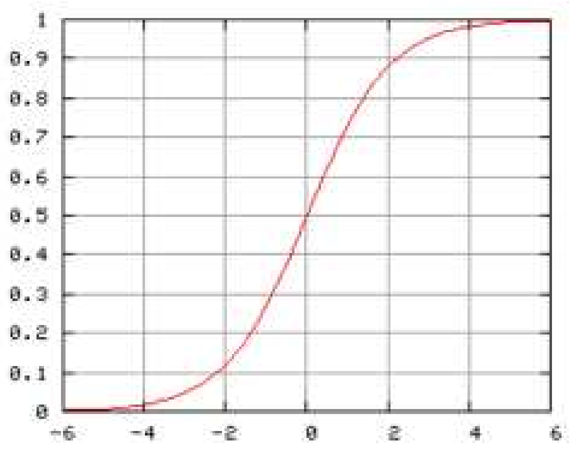


Figure 13: Sigmoid function

As a remark, the sigmoid function can never return “*0*” or “*1*“ due to its asymptotic nature so that values over *0.9* should be treated as *1* and those under *0.1* should be considered as *0*.

Neurons can be grouped into three types of layers: *input*, *hidden* and *output*. The input layer is composed by neurons that represent the data input variables and “feed” next layers of neurons. Next layers, which are sometimes optional, are denominated hidden layers and there may be several of them. The last layer is called output layer, in which each neuron represents an output variable. Each layer is fully connected to the succeeding layer as Figure 14 illustrates.

For linearly separable problems, a sole neuron can categorise the output, but when having more than one class or multimodal spaces at least one hidden layer is needed.

Most statisticians are used to applying regression methods in which data are best–fitted to a specified relationship which is usually linear. However, these methods have several handicaps. For instance, relationships must be chosen in advance and these must be distinguished as linear or non–linear when defining the equation. ANNs enable us to mitigate all these problems.

In regression, the objective is to forecast the value of a continuous variable which is the incoming flow rate in our case. The output required is a single numeric variable which has been normalised between *0* and *1*. ANNs can actually perform a number of regression tasks at once, although commonly each network performs only one.

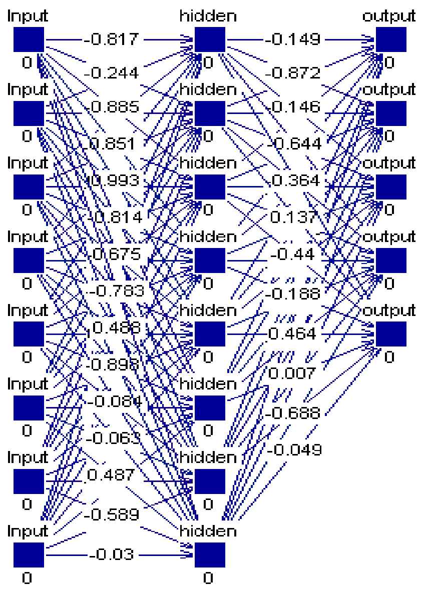


Figure 14: Example of a simple ANN with 8 input neurons, 8 hidden neurons and 5 output neurons, forming 3 fully connected layers.

On the one hand, the main advantage of ANNs is their flexibility to make patterns, being suitable for large and complex datasets as well as long-time-horizon forecasting [68, 71]. On the other hand, we can also find some disadvantages: long computing times, risk of overfitting, need of a feature selection process and difficulty to approach all parameters for each task type [69]. The overfitting, also called overtraining, is the consequence of reducing the error in a specific dataset. When an ANN is trained during a large number of epochs (an epoch is the presentation of the entire training set to the neural network), the function determined by the weights of the ANN may take the particular characteristics of the examples. If this happens, the results will be optimal for the training dataset but no guarantee is given for any other. This risk is minor when the data set is big enough (see Figure 15 [66]).

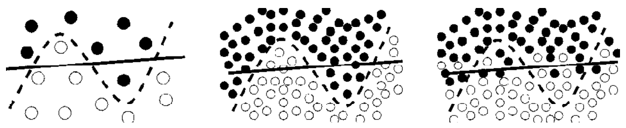


Figure 15: Overtraining risk – Volume of data.

There exists a number of learning algorithms for training ANNs; most of them can be viewed as a clear-cut application of optimisation theory and statistical estimation. They include learning algorithms such as back-propagation by gradient descent [72], back-propagation with momentum [73], resilient propagation [74], quick-propagation [75], Broyden–Fletcher–Goldfarb–Shanno [76], radial basis function [77], Cascade Correlation [78], Hopfield [79], etc.

In this work, we propose an enhanced, self-adaptive gradient-descent based algorithm (an upgraded resilient back-propagation) which is explained in detail in Section 4.3.4.

4.3.2 Background

We have mentioned several learning algorithms for ANNs but we will firstly focus on the back-propagation algorithm (BPA) [72] in order to situate our proposal (if the reader desires further information about other learning algorithms, we recommend reviewing the references given in Section 4.3.1.4).

BPAs can be categorised as multilayer perceptrons [72] which have non-linear activation functions [80] such as the logistic function [81], the softmax function [82], the Gaussian function [83], among others [84, 85]. BPN denotes that any error made by the network when returning an output during the training process is sent backwards with the purpose of correcting it as far as the network learns what is right or what is not. Errors are propagated backwards from output nodes to internal nodes. Therefore, BPA is used to calculate the gradient error of the network with respect to its adjustable weights. This gradient is often used in a simple stochastic gradient-descent algorithm to find weights that minimise that error. BPA simply takes the derivative of the cost function with respect to the network parameters and then changes those parameters in a gradient-related direction. Hence, the most important problem with gradient-descent methods is the premature convergence to local optima which might be far from the global optimum. This problem can be solved by using global optimisation techniques. However, these techniques normally require high computing times.

Nonetheless, other improved gradient-based learning algorithms with more global information such as resilient back-propagation (Rprop) [74] can be more appropriate because the training set is large enough to be effectively applied.

In this context, Rprop is a robust ingrained modification of classical gradient-descent method. This scheme tends to fine-tune an individual step-size to optimise each parameter. The mechanism to perform this action entails doing adaptations of these step-sizes by applying a more or less complex heuristic, instead of considering proportional step-sizes to the partial derivatives. Note that classical gradient-descent algorithms calculate the steepest descent direction by means of an Euclidean metric. Classic Rprop just takes into account the sign of the partial derivative $\frac{\partial \Lambda \varepsilon(t)}{\partial w_{ij}}$ (partial derivative of the error measure with respect the weight between two neurons *i* and *j*) in order to resolve the direction of the weight update. When there is a change of sign of the partial derivative, we can state that a local minimum has been surpassed as there is a change of direction in the search space surface. Besides, we have to update the weights and automatically adapt the step-size, considering the sign the partial derivative. Although we will see how to achieve this weight updating with classic Rprop and our modification in next section, we recommend reading the full description of Rprop algorithm in [86]. This work proposes an upgraded, adaptive modification of the standard resilient back-propagation with weights backtracking (uRprop) learning algorithm.

4.3.3 An Innovative Adaptive Learning Rate Algorithm for Resilient Back-Propagation Neural Networks

This section proposes a modification in weights adjustment for Rprop with weights backtracking in order to make the learning phase more adaptive to environmental circumstances. The main purpose is to properly determine the right weights of our ANN. A great challenge is to find out how big step-sizes (learning rate speed) should be. Note that selecting the right learning rate is always a laborious task.

Rprop basically processes example by example and obtains an output for each one as most ANNs do [73-80]. Each output is compared to the well-known output and this signal is then propagated, calculating the differentials among errors according to the weights (gradients). To update the weights between each pair of neurons *i* and *j* (w_{ij}) after each pattern (epoch, [87]), we inspect the previous weights as follows [87]:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \qquad \textbf{(10)}$$

where $\Delta w_{ij}(t)$ can be formalised like so [74] (classical Rprop):

$$\Delta w_{ij}(t) = \begin{cases} \Delta_{ij}(t), if \left(\frac{\partial \Lambda \varepsilon(t)}{\partial w_{ij}}\right) < 0 \\ -\Delta_{ij}(t), if \left(\frac{\partial \Lambda \varepsilon(t)}{\partial w_{ij}}\right) > 0 \end{cases} \qquad \textbf{(11)}$$

where $\Delta_{ij}(t)$ is the step-size and $\Lambda\varepsilon$ is the error measure (it can be defined as a normalised mean absolute error for the generalisation data set between two consecutive epochs). The weights updating $\Delta w_{ij}(t)$ is carried out until the stopping condition is met. We will employ a stopping criterion based on a fixed number of epochs or a given amount of elapsed time.

Classical Rprop [74] (without weights backtracking) just takes into account the change of sign of the partial derivative (change of direction in the search space surface). This precisely means that a local minimum has been surpassed because the step-size taken has been too long. An important improvement of classical Rprop was to include weight updates, enabling backtracking movements [88]. Weight updating $\Delta w_{ij}(t)$ entails adjusting $\Delta_{ij}(t)$ by applying the following formula:

$$\Delta_{ij}(t) = \begin{cases} \min\left(1.5 \cdot \Delta_{ij}(t-1), \Delta_{max}\right), if \left(\frac{\partial \Lambda \varepsilon(t)}{\partial w_{ij}} \cdot \frac{\partial \Lambda \varepsilon(t-1)}{\partial w_{ij}} > 0\right) \\ \max\left(\frac{\Delta_{ij}(t-1)}{2}, \Delta_{min}\right), if \left(\frac{\partial \Lambda \varepsilon(t)}{\partial w_{ij}} \cdot \frac{\partial \Lambda \varepsilon(t-1)}{\partial w_{ij}} < 0\right) \\ \Delta_{ij}(t-1), other\ case \end{cases} \quad \textbf{(12)}$$

(12) is not of our own (see [89]), we just propose different step-sizes at this point. The key idea is to multiply by *1.5* or divide the step-size by *2*, depending on the surface of the search space. When $\frac{\partial \Lambda\varepsilon(t)}{\partial w_{ij}} \cdot \frac{\partial \Lambda\varepsilon(t-1)}{\partial w_{ij}} > 0$, the signs of the derivates do not change ("+" by "+" or "–" by "–" is always positive). This means that we have not reached the local minimum yet. Therefore, we increase the step-sizes until we have surpassed a local minimum $\left(\frac{\partial \Lambda\varepsilon(t)}{\partial w_{ij}} \cdot \frac{\partial \Lambda\varepsilon(t-1)}{\partial w_{ij}} < 0\right)$. When $\frac{\partial \Lambda\varepsilon(t)}{\partial w_{ij}} \cdot \frac{\partial \Lambda\varepsilon(t-1)}{\partial w_{ij}} < 0$, there is a change of sign in $\frac{\partial \Lambda\varepsilon(t)}{\partial w_{ij}}$. This implies that we have already jumped over the local minimum. When the local minimum is surpassed, we change the sign of the gradient.

Authors [90, 91] typically limit the step-size with $\Delta_{min} = 0$ and $\Delta_{max} = 50$. Instead, we propose $\Delta_{min} = 0.001$ and $\Delta_{max} = 30$ as limits ($\Delta_{min} = 0.001$ for the number precision needed and $\Delta_{max} = 30$ as this already implies a long hop).

Up till now, we have defined how to adjust the step-size $\Delta_{ij}(t)$ at time *t*, depending on the sign of the partial derivative. But, we still have to update $\Delta w_{ij}(t)$. [92] proposes an important improvement to classic Rprop which lies in weight updates with backtracking (reverting a wrong movement or step). Our modification is based on weight backtracking movements but we consider local information of the search space surface by means of the previous error measure ($\Lambda\varepsilon(t-1)$). When there is a change of sign in the partial derivative $\left(\frac{\partial \Lambda\varepsilon(t)}{\partial w_{ij}} \cdot \frac{\partial \Lambda\varepsilon(t-1)}{\partial w_{ij}} < 0\right)$, we calculate $\Delta_{ij}(t)$ as defined in (12). Then, we check out whether the current error measure $\partial\Lambda\varepsilon(t)$ is a *15%* (this percentage can be parameterised) bigger than the previous error measure $\partial\Lambda\varepsilon(t-1)$. When this occurs, we undo the previous movement ($\Delta_{ij}(t-1)$) as we have not only surpassed the local minimum but also gotten a much higher error measure. However, when the deviation is lower than *15%* (we are further from the local optimum but not extremely faraway), we go back to halfway as far as fully reverting a movement leads us to wasting too many iterations. But, if we stay halfway between the previous point of the search space and the current one $\left(\frac{\Delta_{ij}(t-1)}{2}\right)$, the probability of getting closer to the local minimum increases and the total number of iterations needed to reach the local optimum decreases. Hence, we propose to apply the following expressions for each w_{ij} (in order to distinguish what is novelty from what is not, we highlight the code of our own):

$$if\left(\frac{\partial\Lambda\varepsilon(t)}{\partial w_{ij}}\cdot\frac{\partial\Lambda\varepsilon(t-1)}{\partial w_{ij}}>0\right)\left\{\begin{array}{c}\Delta_{ij}(t)=\min\left(1.5\cdot\Delta_{ij}(t-1),\Delta_{max}\right)\\ \Delta w_{ij}(t)=\begin{cases}\Delta_{ij}(t), if\left(\frac{\partial\Lambda\varepsilon(t)}{\partial w_{ij}}\right)<0\\ -\Delta_{ij}(t), if\left(\frac{\partial\Lambda\varepsilon(t)}{\partial w_{ij}}\right)>0\end{cases}\\ w_{ij}(t+1)=w_{ij}(t)+\Delta w_{ij}(t)\end{array}\right\}$$

$$if\left(\frac{\partial\Lambda\varepsilon(t)}{\partial w_{ij}}\cdot\frac{\partial\Lambda\varepsilon(t-1)}{\partial w_{ij}}<0\right)\Bigg\{$$

$$\Delta_{ij}(t)=\max\left(\frac{\Delta_{ij}(t-1)}{2},\Delta_{min}\right)$$

$$if\left(\left(\partial\Lambda\varepsilon(t)>\partial\Lambda\varepsilon(t-1)\right)\&\&\left(\partial\Lambda\varepsilon(t)<1.15\cdot\partial\Lambda\varepsilon(t-1)\right)\right)$$

$$w_{ij}(t+1)=w_{ij}(t)-\frac{\Delta w_{ij}(t)}{2}$$

$$else\ if\left(\partial\Lambda\varepsilon(t)>1.15\cdot\partial\Lambda\varepsilon(t-1)\right) \tag{13}$$

$$w_{ij}(t+1)=w_{ij}(t)-\Delta w_{ij}(t)$$

$$else\ \Delta w_{ij}(t)=\Delta w_{ij}(t-1)$$

$$\frac{\partial\Lambda\varepsilon(t)}{\partial w_{ij}}=0$$

$$\Bigg\}$$

$$if\left(\frac{\partial\Lambda\varepsilon(t)}{\partial w_{ij}}\cdot\frac{\partial\Lambda\varepsilon(t-1)}{\partial w_{ij}}=0\right)\left\{\begin{array}{c}\Delta w_{ij}(t)=\begin{cases}\Delta_{ij}(t), if\ \left(\frac{\partial\Lambda\varepsilon(t)}{\partial w_{ij}}\right)<0\\ -\Delta_{ij}(t), if\ \left(\frac{\partial\Lambda\varepsilon(t)}{\partial w_{ij}}\right)>0\end{cases}\\ w_{ij}(t+1)=w_{ij}(t)+\Delta w_{ij}(t)\end{array}\right\}$$

Note that our learning algorithm may get trapped in local minima but, compared to Rprop, it is faster and usually obtains better results for a reduced amount of training time.

We have defined a generic modification of Rprop with weights backtracking algorithm for ANNs but we still have to formalise the rest of problem-dependent parameters of the ANN (topology activation function, inputs, outputs, hidden units, etc.). Chapter 5 describes how to fine-tune our ANN specifies how to achieve this in a real world environment which is the large call centre of Telefónica.

4.4 SEARCH MODULE

This section describes the key features of the search module which is the second block of the architecture proposed in Figure 7 (second arrow). This block is implemented as a parallel MA. This section contains both own innovations and classical, applicable evolutionary operators (we will not inspect existing evolutionary operators which are not suitable or prevalent for our problem specification). Note that this section is conceived for describing all evolutionary operators and parameters from a generic point of view rather than presenting concrete use case adaptations as Chapter 5 will cautiously put forward how to adjust all parameters to the call centre use case.

4.4.1 Methodology

Once the forecast module (an upgraded resilient back-propagation neural network as described in Section 4.3) has provided us with all the predictions, the search module (a parallel steady-state MA, see Section 4.4.2.5.5) optimises the assignment among task types and agents. The parallel steady-state MA is devised as an island topology (see Section 4.4.4) with migration operators for individuals exchanging, where a master island manages the rest of subordinate islands (note that we do not use the term *slave* as these islands operate with complete solutions and do not merely process partial information). So, each island corresponds to a full steady-state MA.

Each MA keeps a population of abstract representations (chromosomes) of candidate solutions (phenotypes) to the problem described in Chapter 2. The population is partially randomly initialised (for further information, see Section 4.4.2.3). Afterwards, its individuals are evaluated by applying a fitness function over them (we will see an example in Chapter 5). From this population, some individuals are selected and, after that, recombined (crossover). Subsequently, the offspring may suffer perturbations (mutations) in some genes. Then, some of these individuals replace others from the population according to the replacement policy. Every generation includes all previous actions. After that, an LS mechanism is applied over a percentage of the population each *g* generations. Finally, each *mi* generations, the master island halts the slave islands in order to pick up certain individuals from them and spread other ones. All these steps are carried out until a predefined time has been elapsed.

4.4.2 Genetic Algorithm

4.4.2.1 Encoding

The first stage when designing an MA is to define a problem representation (chromosome or genotype) to encode candidate solutions (phenotype) to the problem in a form that every computer can interpret. The "physical" expression of the genotype is called the phenotype. This means that a mapping between genotype and phenotype must be delineated. There are multiple forms to encode candidate solutions which range from binary strings, arrays of integers or arrays of decimal numbers to strings of letters.

Specifically, our solution consists in an integer representation. We just need an array of integers whose indexes represent the available agents, $A_t \subseteq A$, at a given instant, *t*, and the array contents refer to the profile, P_j, assigned to each agent a_i ($<P_1,...,P_i,..., P_l>$). Then, tasks are "routed" to the agents, according to the profiles assigned. Of course, we can also encode the solution as an array of integers whose indexes symbolise the task types and its respective contents represent the number of agents assigned to each task type. This option is recommended whether there are too many agents and hardware capacity is very limited (with respect the total number of available agents). In contrast, we are missing the capability of working at agent's profile level. As we have not this capacity constraint, we will employ the first codification proposed.

Figure 16 shows a fictitious example of encoding (go over Chapter 2 to refresh terminology, if needed) for *10* work items (w_0-w_9) grouped in *3* different tasks types (t_0-t_2) depending on the nature of the tasks, *5* agents (a_0-a_4) and *4* skill profiles (P_0-P_3), where $P_0=\{s_0, s_1\}$, $P_1=\{s_1\}$, $P_2=\{s_2\}$ and $P_3=\{s_1, s_2\}$. Now, suppose that $a_0\sim\{P_0,P_1\}$, $a_1\sim\{P_0, P_2\}$, $a_2\sim\{P_1,P_3\}$, $a_3\sim\{P_2,P_3\}$ and $a_4\sim\{P_0,P_1\}$. We have seen the potential profiles for every agent but only one profile can be assigned to each agent at a given instant *t*; therefore, a feasible solution would be Figure 16 (a_0 and a_4 have been assigned to the profile P_0, a_1 and a_3 have been assigned to the profile P_2, while a_2 has been assigned to the profile P_1).

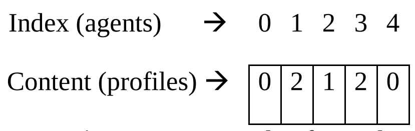


Figure 16: Example of encoding.

4.4.2.2 Population

The population of our MA is a compilation of chromosomes encoded as hinted in Section 4.4.2.1. The population is the minimum unit of evolution since individuals are static elements by themselves. This evolution can be observed in the changes produced in the genetic configuration over the time in each successive generation. The changes between two generations are usually small but these differences mount up with each generation, causing significant changes in the "original" population.

The size of the population often depends upon the nature of the problem and typically contains tens or hundreds of possible solutions. Although there is no rule of thumb to determine the optimal population size, it is recommended to have a population neither too small nor too big, since individuals frequently evolve faster in such an environment.

Now, the concept of diversity must be mentioned. The diversity represents the variety of phenotypes and/or genotypes that a population has. The diversity is essential in a population because the more diverse a population is, the more chances to adapt itself to environmental changes it has.

EAs may have multiple populations that evolve according to rules of the genetic operators. In these cases, a migration operator and a replacement policy are needed.

4.4.2.3 Initialisation

Typically, the initial population is fed with randomly generated individuals who should potentially cover different possible configurations. In some cases, we can use other algorithms to initialise the population (e.g. a more or less complicated LS mechanism) but, in most cases, this is not possible since computing times increase too much and real applications require short computing times. In our case, we propose to start from a random initial population, including the best solution found in the previous time-frame because the configuration of agents' profiles should not change too much over two successive time-frames.

4.4.2.4 Fitness Function

The fitness function is an evaluating mechanism which is defined over the chromosome to measure the quality of a given solution. This function often guides the search and decides which individuals must be selected for the next generation (in fact, surviving individuals also depends on the replacement policy). The fitness function is intrinsically linked to the problem. Frequently, the hardest action when defining an EA is to identify the right fitness function since results strongly depend on it. Occasionally, it is hard (sometimes impossible) to characterise the fitness expression; in these cases, interactive genetic algorithms are used. In other cases, long evaluating times imply that an approximate function is needed. The fitness function is problem dependent and Chapter 5 will carefully describe a fitness function for a multi-skill call centre. Besides, we will explain how to incorporate constraints to a dynamic environment as already stated in Chapter 2. We will also propose a mechanism to calculate a partial fitness function instead of recalculating everything in each evaluation (as this is problem dependent, we cannot include this mechanism in this section).

4.4.2.5 Evolutionary Operators (Classic and New Operators)

In this subsection, we explain potentially appropriate evolutionary operators which may be applied to the problem described in Chapter 2, given the encoding we are proposing in Subsection 4.4.2.1. This section does not attempt to cover all feasible evolutionary operators, just the ones we consider relevant for DMAS. Some of these evolutionary operators are innovative but others are not (this is specified for each one).

4.4.2.5.1 Selection Operator

Since the population needs to be bred each successive generation, several individuals are chosen to be recombined. In the state-of-the-art, one can find the following ones:

Random Selection: consists in randomly selecting a configurable percentage of individuals for potentially recombining them.

Tournament selection [93]: implies executing *t* tournaments among some individuals randomly chosen from the population. The individual who has the best fitness is selected for recombination. When *t* is larger, individuals with worse fitness have fewer chances to be selected (*t* indirectly determines the selective pressure).

Roulette-wheel selection [28]: associates a probability of selection with each individual chromosome. The probability of selecting a chromosome is proportional to its fitness or rank (survival of the fittest).

Truncation selection [94]: removes a predetermined percentage of the candidates with worst fitness.

Ordered selection [95]: randomly picks a chromosome from the top *N* percent of the population.

Best: merely selects the best chromosome in terms of fitness. When there are more than two chromosomes with the same best fitness, one of them is randomly chosen.

4.4.2.5.2 Mating Operator

The purpose of this operator is to mate individuals (which individual should reproduce with another one). We can hit upon the following techniques:

Random mating [96]: randomly mates individuals for posterior crossover.

Fitness-based mating [97]: selects pairs of individuals with the highest difference in terms of fitness (best fitness individual will be mated with the worst fitness one). The idea is to potentially provide the EA with a fast diversity mechanism.

Similarity mating [98]: selects pairs of individuals having more differences in terms of genes in their chromosomes. This mechanism provides real diversity to the EA but it is time-consuming.

4.4.2.5.3 Crossover

This operator combines individuals to produce several children (offspring). The key idea behind the recombination of individuals is to potentially obtain other better fitted individuals.

One-point crossover [99]: chooses a random point on both parents' chromosomes (the same point for both parents). All the genes until this point from one of the parents (randomly chosen) are copied to one of the children (randomly chosen). The genes beyond this point from the first father are arbitrarily copied in the other child and the ones from the second father are arbitrarily copied in the other child as Figure 17 illustrates.

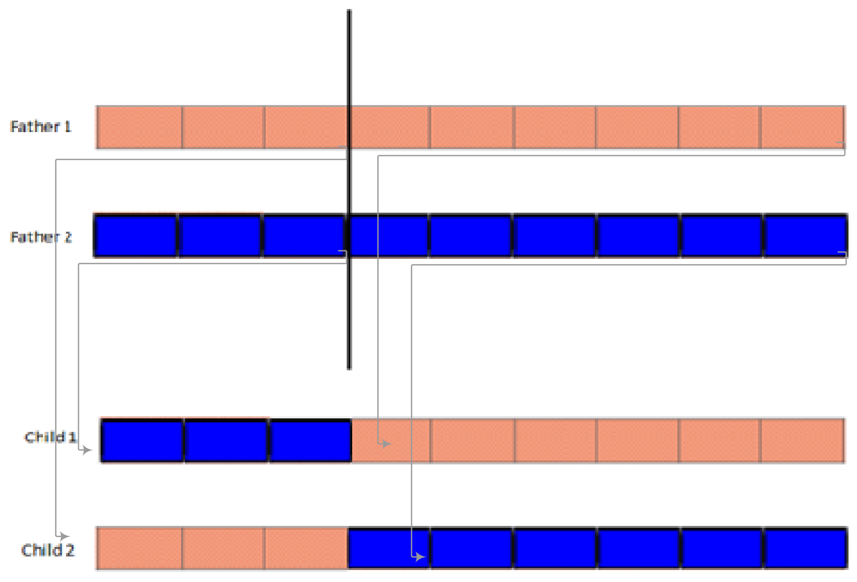


Figure 17: One-point crossover.

Multi-point crossover [100]: selects *N* random points on both parents' chromosomes (the same points for both parents). Each piece of chromosome from the parents is alternatively copied in each child as Figure 18 shows.

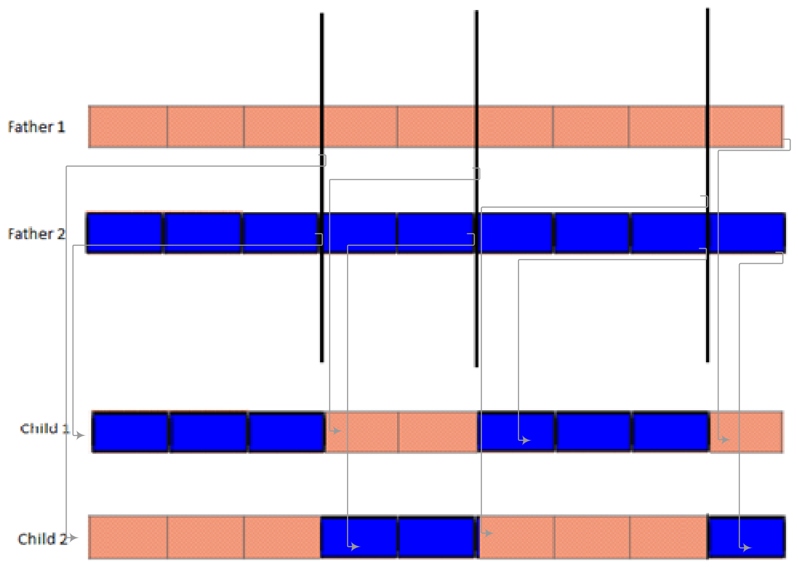


Figure 18: Multi-point crossover.

Cut and splice [28]: consists in selecting *2* different random points (one in each parent). One piece of father-*1*'s chromosome is then copied in a randomly chosen child. The same action is accomplished for father-*2* in the opposite child. The rest of genes are randomly copied (see Figure 19).

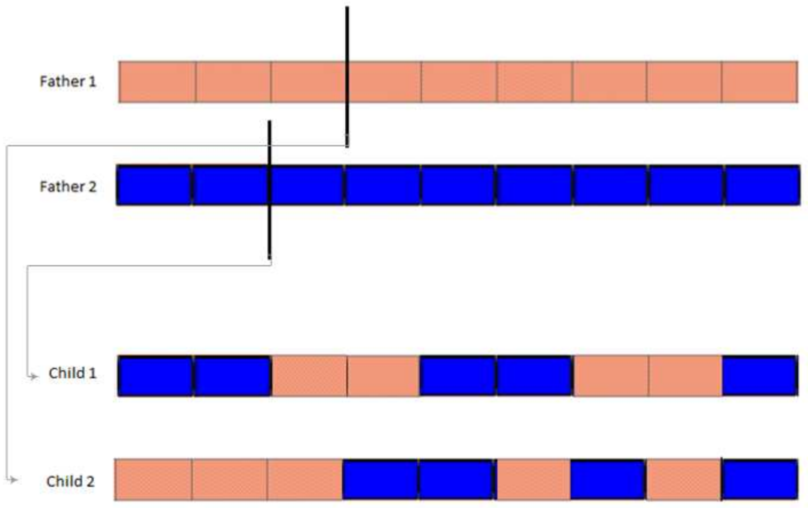


Figure 19: Cut and splice.

*Probability crossover***:** considers that children will inherit the common points in their parents (potentially, the best genes) and randomly receive the rest of genes from them. This probability can be the *0.5* (uniform crossover [101]) or proportional to the fitness.

We also propose to assign this probability in a more complex way such as simulated annealing does [45]. At the beginning of the process, when the temperature is higher, we can explore more by applying a probability of *0.5* and, when the temperature starts cooling off, we can give more probability to the best fitted individual as follows:

Pr_0 *(best fitted) = 0.5* (initial probability for best fitted parent).

Pr_i *(best fitted) = 0.5 + v/T_i* (probability for the best fitted parent at generation *i*). **(14)**

where T_i is temperature at iteration *i* (the probability of giving a higher weight to the best fitted individual increases when the temperature decreases) and *v* is a factor to return values between *0* and *0.5*.

Simulated annealing has different schemes to decrease the temperature but they all decrease nonlinearly. Another option is to increase the probability according to the number of generation generated as follows:

$$Pr_i\,(best\ fitted) = 0.5 + v/G_i \qquad \textbf{(15)}$$

where G_i stands for the generation number i and *v* is a factor to return values between *0* and *0.5*.

All in all, the idea is to choose a probability for recombination and we have several mechanisms to achieve this task as Figure 20 confirms:

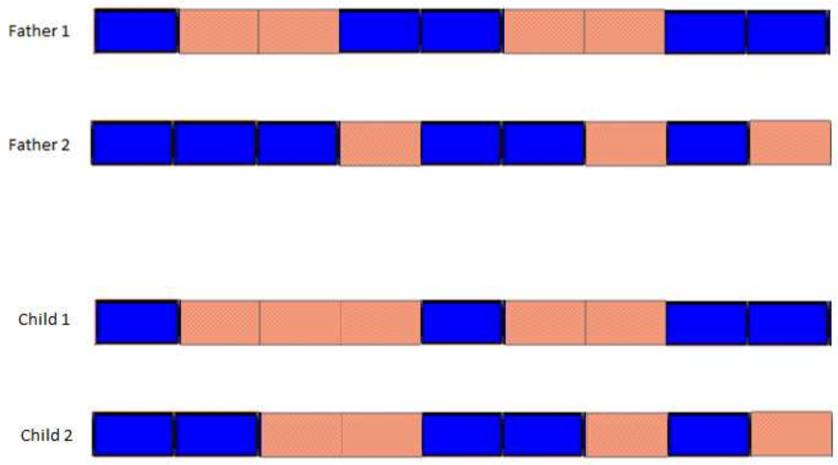


Figure 20: Probability crossover.

4.4.2.5.4 Mutation Operator

This operator causes tiny changes in the chromosome of individuals to explicitly maintain diversity. It applies a perturbation over each gene of the chromosome with a given probability. This perturbation corresponds to changes of profiles in some agents (e.g. agent a_2 who had assigned the *profile* P_1 has now associated the *profile* P_3 due to a mutation).

4.4.2.5.5 Replacement Policy

Finally, we decide which individuals are incorporated (or maybe reinserted) into the population for the next generation.

Generational [102]: After recombination, the offspring generated by the selected parents fully replaces them. The selection strength is low when this scheme is applied (slow convergence). However, it potentially converges to the global optimum when enough generations are generated.

Steady-state [102]: After crossover, the offspring generated by the selected parents may replace them if these are best fitted. There are numerous policies for individuals' replacement:

- *Elitism* [103]: best fitted individuals fully replace the worst ones (quick convergence).
- *Random replacement*: randomly chooses the individuals from the parents and children set. With a probability, worst fitted individuals may replace the best ones.
- *Boltzmann criterion* [45]: The best fitted individual is chosen and another one (which may not be the second best fitted individual) is inserted with a given probability as simulated annealing does.
- *Similarity criterion*: we propose to select the best fitted individual and its most different one in terms of genes.
- *Taboo criterion*: we also propose another scheme which lies in storing a list of non-promising individuals (based on their age for instance) in order to avoid inserting duplicated or inappropriate individuals. This option imposes

additional memory requirements and more evaluations. Instead, we save up so much time in incorporating useless individuals during a given amount of time or iterations.

4.4.2.5.6 Stopping Condition

Elapsed time: considers a fixed amount of time to run the algorithm.

Number of generations: executes a predefined number of generations.

Number of generations without an improvement in fitness: executes the algorithm until there is no improvement (or just under a given threshold of upgrading) during the latest *g* generations. Another choice is to keep the best solution found and restart the algorithm, employing another algorithm configuration (parameterisation) in order to find a better optimum in other place of the search space when the stopping criterion is one of the previous ones (if we still have time or remaining generations, we better utilise this time or generations in searching other possibilities).

4.4.3 Memetic Algorithms

As we described in Section 3.1.3, MAs are a population-based technique for heuristic search in optimisation problems. MAs are quicker than traditional GAs for many problem domains because these apply an LS procedure. The present section describes the MA we propose.

4.4.3.1 Local Search

LS is an MH for solving optimisation problems. An LS algorithm starts out from a candidate solution and, thus, iteratively moves to a neighbour solution, generating the neighbourhood. To carry out this action, a neighbourhood relation must be defined on the search space. In our case, we state that two candidate solutions are neighbours if only one gene differs in both chromosomes. Note that we propose a “simple” LS due to the lack of time of a dynamic environment but a more complex LS mechanism may be used when computing times are more flexible.

The following pseudo-code illustrates the LS algorithm:

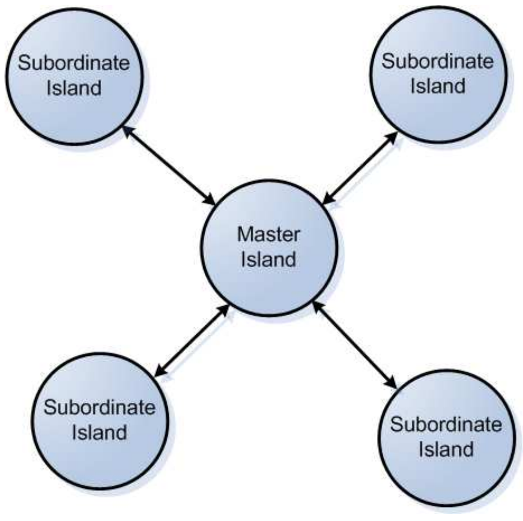
```
void Local_Search (Chromosome & candidate_solution)

   Chromosome best_solution = candidate_solution;

   Chromosome neighbour = candidate_solution;

   For (i=0; i<candidate_solution.size(); i++)

        Agent a = neighbour.getAgent(i);
        For (j=0; j<a.get_number_profiles(); j++)

              neighbour.change_profile(i,j); //profile j for agent i

                 If (neighbour.fitness() > best_solution.fitness())  best_solution = neighbour;

        neighbour = best_solution;

   candidate_solution = neighbour;
```

Another relevant task is to decide the right frequency which should be considered to apply the LS over the population and how many individuals must be affected. Chapter 5 will suggest an LS frequency and a percentage of affected individuals for the multi-skill call centre use case.

4.4.4 On Parallelising Memetic Algorithms

There are many approaches to MAs parallelisation. Due to the constraints in the number of pages for this document, we are not going to go over all of them and will just describe the configuration we propose for this type of dynamic environments.

We propose an *island model* where there are a master island and *s* subordinate islands. Every subordinate island is connected with the master one but not with the others. The master island asynchronously stops the rest of subordinate islands and asks for a percentage of their best fitted individuals. Then, the master island takes these best fitted individuals and decides whether (or not) to incorporate these individuals into its population. Then, the master island sends back its best *N* individuals and the most different one to the best fitted one in terms of genes differences. Subordinate islands apply elitism to accept or not the incorporation of these individuals (whether the individuals coming from the master island are not better fitted than the existing ones). The process is carried out until the stopping condition is met.



CHAPTER 5. APPLICATION: CALL CENTRE

In Chapter 5, we adapt our approach to a real-world DMAS: the multi-skill call centre. Section 5.1 describes the specific characteristics of our problem domain. Section 5.2 presents a brief survey of call centre algorithms. Section 5.3 highlights the magnitude, in terms of volume, of our application domain. In Section 5.4, we present some special adaptations for the forecast module. In contrast, Section 5.5 points out some particular adaptations for the search module.

5.1 DESCRIPTION

A call centre (CC) [11] is a centralised office used for receiving and transmitting large volumes of telephone requests which may range from customer service to the selling of products and services. Even though CCs have been broadly studied, there are still some lacks on optimisation which may imply huge losses of money every year because of a wrong allocation of resources to the right tasks, and client dissatisfaction due to never-ending delays as pointed out in Section 1.4 (Market Relevance).

In a CC, the flow of calls is often divided into outbound and inbound traffic. *Outgoing calls* are handled by agents, primarily, with commercial pretensions. This type of calls is planned as agents know in advance which customers must be contacted every day. Conversely, *incoming calls* are those that go from the client to the CC to contract a service, ask for information or report a problem. These unplanned calls are initially modelled and thus classified into manifold call groups (CGs) in relation to the nature of each call (complaints, V.I.P. clients, client loyalty, etc.). As soon as these CGs have been modelled, each call is assigned to a unique CG (there is no overlap among CGs). Each incoming call needs time to be answered, requiring different processing times as indicated below:

1) The first one is the time needed to assign a type to the call (modelling).

2) The second one is the time that the call is queuing (waiting).
3) The last one is the time that the agent needs to handle the call (processing).

A key component for any CC is the automatic call distributor (ACD) which is a system that models incoming calls and automatically distributes them throughout different queues from which certain agents can pull work. The routing scheme is a rule-based set of operations that guides the ACD to handle a given incoming call inside the system. Typically, once the call has been assigned to a queue, a second algorithm is required to select the best available agent to reply to a given incoming call.

Habitually, the distribution of the incoming flow is based on the current state of the queues. A CC is a changing environment where conventional algorithms have no time to reach an optimal solution. It would be desirable to predict the future state in order to give more time to the algorithms to consider the "whole picture" of the situation to efficiently reallocate every agent. The majority of traditional techniques is supported by a strong assumption which relies on the way that incoming traffic arrives. Most techniques suppose that incoming flows within CCs follow a Poisson distribution. In this context, the main concern should be to forecast, for an upcoming state, the inbound traffic, abandonment rate and available agents having the required skills, in order to properly divvy up the workload among agents as our resources can be, at this point, optimised by a search algorithm. Bear in mind that a fair allocation of workforce improves client satisfaction and, furthermore, reduces costs.

A specific type of CC is the multi-skill call centre (MSCC). In an MSCC, there are n customer calls grouped in k types of calls and m agents that may have up to l skills ($l \leq k$). This implies that each agent can attend different types of calls and, given a type of call, it can be answered by several agents that have that skill.

Obviously, the scenario can be simpler in some special CCs in which agents have a single skill. These CCs can be modelled with q single queues working in parallel. In other cases, every agent has all possible skills; hence all customers are queued in a single queue that can be handled by any agent. The system is noticeably easier to analyse in these two extreme cases. With all agents having all skills, the system is also more efficient (shorter waiting times, fewer abandonment rates) when the service time distribution for a given call type does not depend on the agent's skill set. However, this assumption turns out to be wrong in practice: agents are usually faster when they handle a smaller set of call types (even if their training gives them more skills). Agents with more skills are also more expensive as their salaries depend on their skill sets. Thus, for large volumes of call types, it makes sense to dedicate a number of single-skill agents (specialists) to handle most of the load. A small number of agents, proportional to the calls of each type, with two or more skills can cover potential fluctuations in the arriving load. To address these fluctuations, the skills are grouped in skill profiles (subsets of skills) so that we can assign an agent to specific types of tasks during a given period of time, despite this agent has skills to process other types of work.

As it can be expected, the mean arrival rate is not the same for each CG as well as the calls of these CGs have different processing times. Now, bear in mind that inbound flow in CCs is usually not a stationary Poisson process [104, 111] and, the service times do not increase exponentially as explained in Section 4.3.1.1. Since calls randomly arrive according to a stochastic process, it would be desirable to have a well-balanced allocation of the agents, who can be available or not, in order to handle the calls as soon as possible.

Figure 21 illustrates the relationship among client calls, queues and agents. This figure describes an example for *9* client calls grouped in *4* CGs and *5* agents having different real skills.

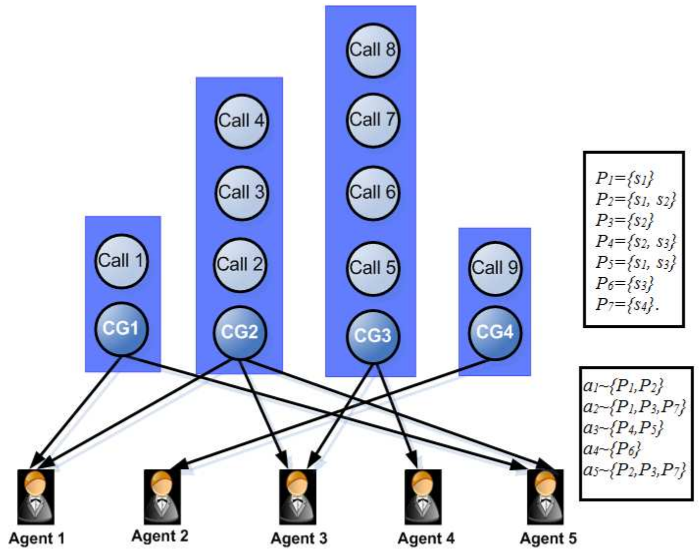


Figure 21: Inbound scheme.

More formally speaking, the following parameters can be found in an MSCC:

1) a finite set of *n* customer calls $C = \{ c_1, c_2, \ldots, c_n \}$.

2) a finite set of *k* CGs (call groups/types) $CG = \{ cg_1, cg_2, ..., cg_k \}$, where $k \leq n$ when every CG has, at least, one call queuing.
3) a finite set of *m* agents $A = \{ a_1, a_2, ..., a_m \}$. Usually, *m >>k*.
4) a finite set of *k* agent-skills $S = \{ s_1, s_2, ..., s_k \}$ in which each agent-skill, s_i, represents the ability to handle the associated CG, cg_i, with the corresponding sub-index in CG: $s_1 \sim cg_1, s_2 \sim cg_2, ..., s_k \sim cg_k$.
5) a finite set of *d* agent-skill profiles $P = \{ P_1, P_2, ..., P_d \}$ in which each agent-skill profile *P_i* can be any subset of $S = \{ s_1, s_2, ..., s_k \}$.
6) a finite set of *n* operations (execution or processing of each customer call, c_i) $O = \{ o_1, o_2, ..., o_n \}$ in which each operation, o_i, has associated a processing time which depends on its CG: $\{\tau_1, \tau_2, ..., \tau_k\}$.

The solution to the problem of the workforce distribution in MSCCs is defined as the right assignment for every agent a_i to the most suitable skill profile P_j from his/her real skill profiles for each *v* seconds, where *v* is the size of the time-frame considered.

In addition, the assignment $\langle a_i, P_j \rangle_t$ must satisfy all hard constraints and handle the soft ones given by the business units. To determine whether (or not) a given solution is suitable, we need to define a quality metric to evaluate the rightness of each feasible solution. There are very significant metrics to measure the quality of a CC such as the abandonment and service rates. These metrics somehow hinge on the (customer) service level [12] which is defined as the percentage of customer calls that have to queue shorter than a specified amount of time. Our work has been conducted by applying this metric.

Moreover, the solution must fulfil the following descriptions:

1) on *O* define *R*, a binary relation which represents the precedence among operations. If $(o_1, o_2) \in R$ then o_1 has to be performed before o_2.
2) each agent, a_i, has associated a finite non-null subset of P, containing his skills to handle different customer CGs (individual real skill-profile).
3) the same profile P_i can be assigned to several agents. In other words, several agents may have some skills in common (or even all of them).

4) every agent, a_i, may have several profiles assigned but only one can be performed at a given instant t, $\left\langle a_i, P_j \right\rangle_t$. In other words, an agent cannot process two (or more) incoming calls at the same instant.
5) every solution must respect diverse (hard and soft) constraints given by business rules defined by business units or agents' regulations.

Likewise, an initial step to produce a planning is to predict future system loads, comprising predicted arrivals, existing queuing calls, abandonments and mean service times. Intuitively, the mean arrival rate for each CG is not the same and their calls may involve different processing times. Note that incoming flow in CCs is usually not a stationary Poisson process and, the service times do not increase exponentially. Since calls randomly arrive according to a stochastic process, a well-balanced distribution of agents is needed with the aim of handling calls as soon as possible.

The complexity of this problem is huge because we are not only dealing with an NP-hard problem like in the job assignment problem, but also considering high dynamism, massive incoming customer calls and large number of agents having multiple skills. Besides, since customer calls are not planned, this makes the call assignment a truly laborious task.

5.2 BRIEF STATE-OF-THE-ART ON CALL CENTRE ALGORITHMS

Reviewing the state-of-the-art, one can realise that many algorithms for workload distribution in single-skill CCs are available (e.g. [105]) because, in the past, agents were commonly allocated to single customer call groups. Nevertheless, not much work has been conducted to workload distribution in MSCCs which is the emblematic scenario in nowadays CCs. In the rest of this section, we discuss the main contributions to workforce distribution in MSCCs.

Workload distribution in MSCCs has been broadly faced by an SBR algorithm [106]. SBR is a call-assignment strategy used in CCs to assign incoming customer calls to the most suitable agent, instead of simply choosing next existing agent. The need for SBR has arisen, as CCs have become larger and deals with a wider variety of call types. The major handicap of this approach is that online (ad-hoc) routing heuristics cannot be very complex in view of the fact that a very short response time is required. These fast, unplanned decisions may imply suboptimal task types assignments to existing agents.

Conversely, Thompson [107] proposes an integer programming model which differentiates minimum acceptable service levels per time-frame from a constraint on the mean service level over the planning horizon. Although this approach considers prospective situations, it is less dynamic to changes than SBR.

Other approaches consider dependent planning intervals (e.g. [108]). Most methods perform well enough within separate intervals but their performance decreases when moving to the next one, giving much trouble in prospective time-frames.

Other authors take into consideration overflow routing in multi-skill blocking systems with randomisation parameters by applying a branch-and-bound algorithm (e.g. [109]) or cutting planes (e.g. [110]). These techniques are only appropriate for stable environments because they need long response times and their performance highly decreases in large instances.

Finally, we can find one of the most representative algorithms of the state-of-the-art (Koole et al., 2008 [11]). Koole presents a heuristic, which considers the costs of agents and a service-level condition, to optimise the distribution of agents among different CGs. This algorithm is faster than most of the aforementioned approaches but deals with specific types of MSCCs in which customer calls arrive according to a Poisson process with deterministic rate. However, note that inbound flow in MSCCs is usually not a stationary Poisson process [104, 111] and, the service times do not increase exponentially. Since calls arrive randomly according to a stochastic process, agents must be well-distributed to handle the calls as soon as possible. Besides, the previous techniques often consider a high granularity and need to work at agent groups' level instead of an agent's profile level. This setback does not enable us to offer more accurate configurations for DMAS.

To conclude, we have seen, in this section, how some approaches employ "basic" heuristics to dynamically distribute incoming customer calls to agents while others cope with stable inbound flows and longer stability over time. In this context, a large-time-frame planning cannot be carried out because of the continuous changeability of all variables involved. Moreover, "basic" heuristics based on the current situation (online routing strategies) may work under certain cases, e.g. stable workload, but daily use of these techniques will guide us to appalling solutions.

5.3 MAGNITUDE OF OUR CALL CENTRE

This section presents some numbers to realise of the magnitude of our CC:

1) Maximum number of CGs = *1.035*
2) Maximum number of simultaneous incoming calls = *2.500*
3) Maximum number of incoming calls per hour = *60.000*
4) Maximum number of incoming calls per day = *700.000*
5) Minimum number of simultaneous agents = *0*
6) Maximum number of simultaneous agents = *2.100*
7) Minimum number of agents concurrently assigned to a single group = *0*
8) Maximum number of agents concurrently assigned to a sole group = *526*
9) Mean number of agents concurrently assigned to a single group = *3*
10) Minimum number of potential profiles per agent = *1*
11) Maximum number of potential profiles per agent = *108*
12) Mean number of potential profiles per agent = *16*

Obviously, the number of incoming calls is not the same all the time as it depends upon many factors. When agglomerating many data and considering a coarse-grain, forecasting becomes much easier as the variability at high level (e.g. monthly and daily level) is reduced and easy to forecast. However, our predictions rely on a fine-grain process as forecasts refer to each successive state. We can perceive that fact in the following figures (Figures from Figure 22 to Figure 25).

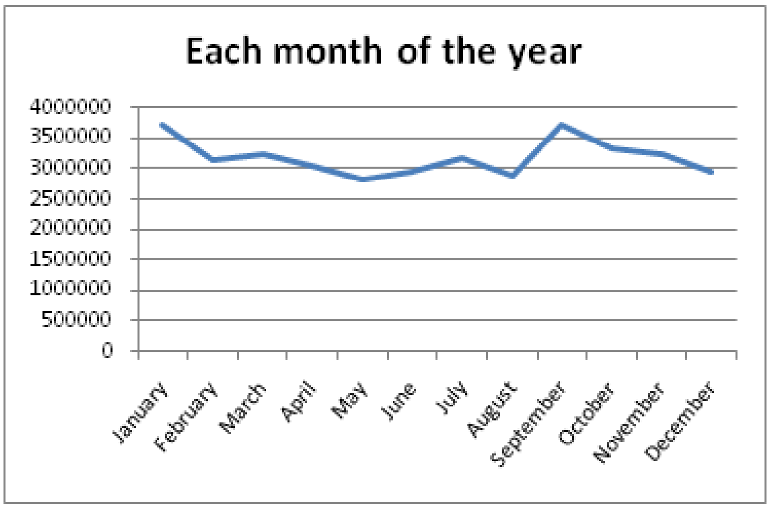


Figure 22: Incoming calls during a year at monthly level.

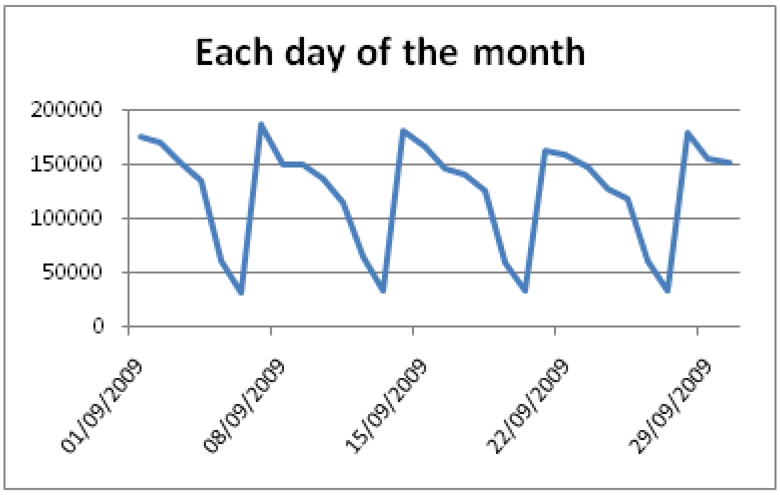


Figure 23: Incoming calls during the most intricate month (September) at daily level.

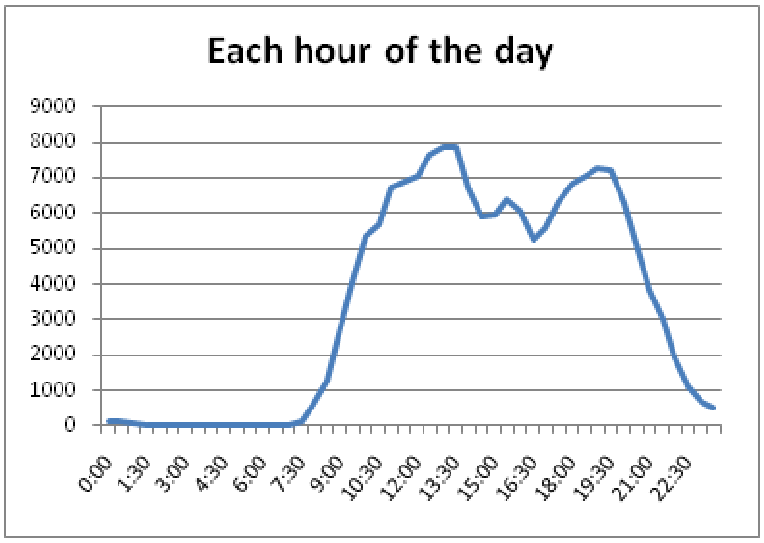


Figure 24: Incoming calls during the most complex day of September (September 9) at hourly level.

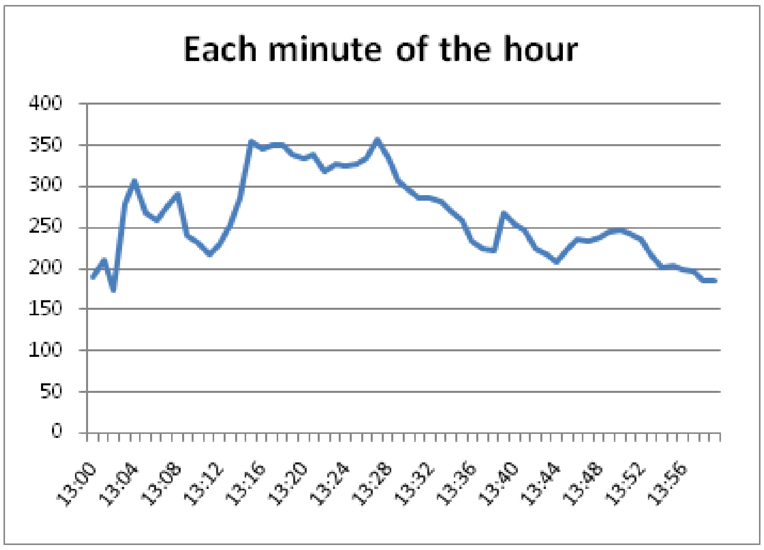


Figure 25: Incoming calls during the most complex hour of “September 9” at minutely level.

A pattern can be relatively easily found at macroscopic level (month of the year and day of the month). However, predictions are harder when considering the hour of the day and much harder at minutely level (our concern in this work). Thus, these figures corroborate how complex is to predict the incoming flow in our environment. However, abandonments and available agents are easier to forecast because:

1) the abandonment rate is highly correlated to the volume of incoming calls as Figure 26 illustrates,

2) and the number of available agents can be inferred from timetables and mean processing times as well as current load and other well-known factors. Figure 27 shows the volume of existing agents as a total value and separating the most representative CGs.

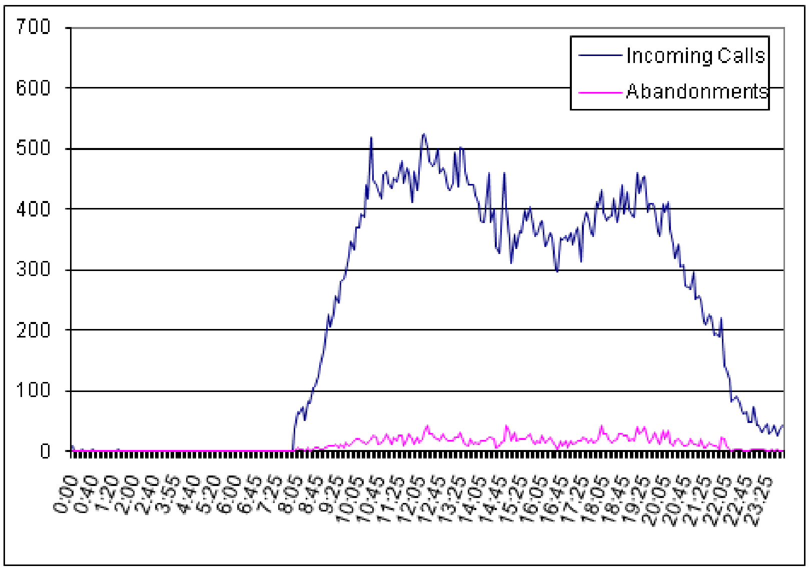


Figure 26: Incoming calls and abandonments during a common day.

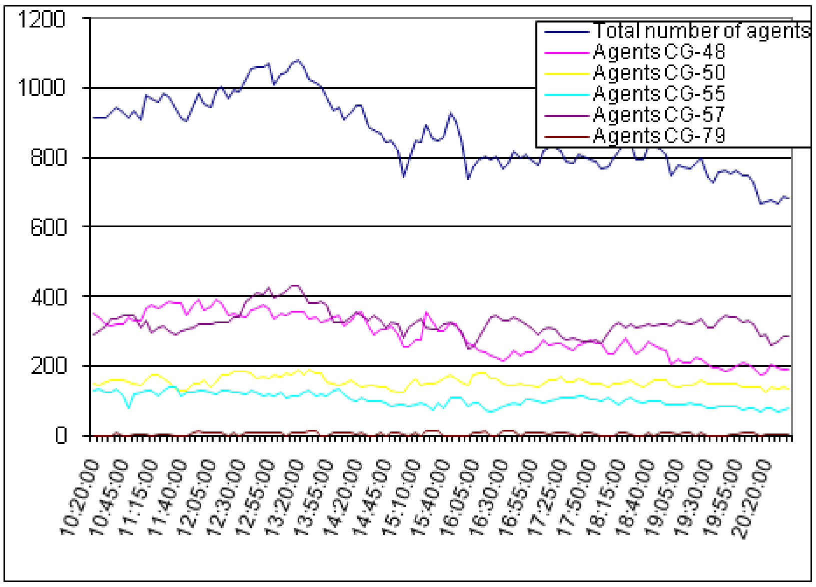


Figure 27: Number of agents for the *5* most representative CGs.

5.4 FORECAST MODULE ADAPTATIONS

This section elucidates the required tuning to adapt the forecast module described in Section 4.3 to the CC's environment. Note that our forecast module relies on an ANN based on uRprop. Our uRprop learning algorithm does not actually need any specific adaptation to our environment (the neuron activation function and the weighted sum of coefficients are also the same ones that we described in Section 4.3) but the architecture of the ANN demands some additional tuning. Our focal control over this architecture relies on the number of hidden layers as well as the number of neurons in these layers because the number of input/output neurons is determined by the number of inputs and outputs we have.

As the number of available agents mainly depends on the agent timetable and the number of abandonments is proportional to the number of incoming calls, we will mainly focus on the prediction of incoming calls. The following sections justify the pertinent configuration of our ANN for the MSCC's domain.

5.4.1 Number of Layers

As claimed in section 4.3, the number of layers of an ANN must be, at least, two (*1* input, *h* hidden where $h \geq 0$ and *1* output). Sometimes, the hidden layer is not needed (e.g. simple linearly divisible problems). In our case, we propose three layers: *1* input layer, *1* hidden layer and *1* output layer. Note that we necessitate a hidden layer at least because our problem is nonlinear. Nevertheless, we do not in fact need more than a hidden layer because we can approximate well enough every function by utilising a single hidden layer with an arbitrarily large number of hidden units (universal approximation property [112]). Of course, the more hidden layers we have, the more accurate our prediction might be (more coefficients in the global formula of the ANN). But, this increases the computing time to train the network (more loops to update the weights) and bear in mind we have limited time to accomplish this task. Besides, adding more than a hidden layer aggravates the problem of getting trapped in local minima [87].

5.4.2 Input Layer

The number of neurons of the input layer is determined by the number of variables we have. But, what variables or features do we have in our environment? At a typical MSCC, we can stumble on a wide variety of variables which may range from information of previous calls (number of calls, tendencies, mean processing times, etc.) to contextual information (campaigns, peak hours, night shift timetable, etc.). We can directly take this raw information but, sometimes, we have too many variables or features to take them into account. As we have much information and many dimensions (variables), it makes sense to reduce the number of input variables. Moreover, sometimes, some variables may even inject identical information into the ANN because of the dependency among variables. To mitigate this drawback, feature selection appears as a promising solution. *Feature selection* is the technique, broadly applied to machine learning, of selecting a subset of relevant features or variables in order to build robust learning models.

Choosing the right inputs from all information we have (*122* different variables) is not trivial and is very important for obtaining a higher performance as having more predictors implies adding new dimensions to the model (more complexity). Since variable selection should not be defined ad-hoc, principal component analysis (PCA) [113] has been employed (see Table 4). PCA is a statistical technique that converts a set of potentially correlated predictors into a smaller subset of uncorrelated predictors designated as principal components. The main advantage of PCA is the capability to compress data by reducing the number of dimensions without significant loss of information. To select the right inputs, we have compiled a dataset of *3* months and obtained the results given by Table 4 (first column stands for the relevance, second column refers to the component number and third column is the component itself).

As a remark, PCA has not been implemented by the author; instead, we have used the Weka framework [114] in order to determine the right inputs for our ANN (see Table 4 and Table 5).

Table 4: Ranked attributes.

Relevance	#	Component
0.8514	1	0.182i_TENDENCIA_9+0.181i_TENDENCIA_18+0.179i_TENDENCIA_8+0.176i_TENDENCIA_17+0.167i_TENDENCIA_7...
0.7635	2	0.293i_LL_TOTALES_5+0.293i_LL_TOTALES_6+0.289i_LL_TOTALES_7+0.289i_LL_TOTALES_4+0.284i_LL_TOTALES_3...
0.6887	3	-0.213i_TENDENCIA_13-0.191i_TENDENCIA_12-0.19i_TENDENCIA_14-0.178i_TENDENCIA_4-0.176i_TENDENCIA_10...
0.6193	4	-0.245i_TENDENCIA_21+0.221i_TENDENCIA_11+0.221i_TENDENCIA_10-0.219i_TENDENCIA_22-0.205i_FLAG_21...
0.5563	5	-0.245i_TENDENCIA_33-0.207i_TENDENCIA_28-0.206i_FLAG_33+0.2 i_TENDENCIA_43+0.199i_TENDENCIA_20...
0.4979	6	0.265i_TENDENCIA_38-0.26i_TENDENCIA_27+0.227i_FLAG_38-0.223i_FLAG_27-0.196i_TENDENCIA_45...
0.4413	7	0.288i_TENDENCIA_26+0.244i_FLAG_26-0.207i_TENDENCIA_32-0.196i_TENDENCIA_33-0.195i_TENDENCIA_3...
0.3861	8	0.257i_TENDENCIA_41+0.217i_FLAG_41-0.208i_TENDENCIA_45-0.205i_TENDENCIA_37+0.191i_TENDENCIA_7...
0.3323	9	-0.271i_TENDENCIA_1-0.233i_FLAG_1-0.202i_TENDENCIA_2-0.197i_TENDENCIA_6-0.19i_TENDENCIA_4...
0.2805	10	-0.34i_TENDENCIA_46-0.291i_FLAG_46+0.283i_TENDENCIA_44+0.243i_FLAG_44+0.202i_TENDENCIA_35...
0.2471	11	-0.45i_INTERVALO_8_2-0.376i_MINUTOS_DIA+0.328i_INTERVALO_8_1-0.323i_INTERVALO_4_4-0.304i_HORA_PUNTA2...
0.2232	12	0.416i_INTERVALO_8_0+0.416i_NOCTURNO-0.313i_INTERVALO_8_1+0.308i_INTERVALO_4_0+0.265i_INTERVALO_4_1...
0.2098	13	0.571i_INTERVALO_4_5-0.451i_INTERVALO_4_4-0.432i_HORA_PUNTA2-0.184i_DIA_SEMANA_6+0.181i_MINUTOS_DIA...
0.1995	14	0.432i_INTERVALO_4_3-0.414i_INTERVALO_4_2+0.244i_DIA_SEMANA_6+0.204i_DIA_SEMANA_5-0.186i_DIA_SEMANA_2...
0.1898	15	0.821i_DIA_SEMANA_2-0.447i_DIA_SEMANA_3-0.197i_DIA_SEMANA_1+0.166i_DIA_SEMANA_6-0.15i_DIA_SEMANA_4...
0.1802	16	-0.724i_DIA_SEMANA_4+0.524i_DIA_SEMANA_3+0.349i_DIA_SEMANA_6-0.164i_DIA_SEMANA_5+0.112i_INTERVALO_4_1...
0.1706	17	0.725i_DIA_SEMANA_5-0.478i_DIA_SEMANA_4-0.423i_DIA_SEMANA_6+0.159i_DIA_SEMANA_0-0.151i_DIA_SEMANA_3...
0.1611	18	0.685i_DIA_SEMANA_1-0.512i_DIA_SEMANA_3+0.313i_DIA_SEMANA_6-0.222i_DIA_SEMANA_4-0.207i_DIA_SEMANA_5...
0.1516	19	0.828i_DIA_SEMANA_0-0.487i_DIA_SEMANA_1-0.188i_DIA_SEMANA_3+0.104i_DIA_SEMANA_6-0.101i_DIA_SEMANA_2...

0.1428	20	-0.689i_INTERVALO_4_1+0.564i_INTERVALO_4_0+0.191i_INTERVALO_4_2-0.181i_INTERVALO_4_3-0.158i_DIA_SEMANA_4...
0.1351	21	0.443i_INTERVALO_4_2-0.364i_INTERVALO_4_3-0.32i_INTERVALO_4_0+0.26 i_INTERVALO_4_1+0.182i_HORA_PUNTA...
0.1289	22	0.332i_FLAG_10+0.332i_FLAG_11-0.169i_TENDENCIA_11-0.169i_TENDENCIA_10-0.16i_HORA_PUNTA...
0.1233	23	0.41 i_FLAG_11+0.41 i_FLAG_10-0.186i_TENDENCIA_11-0.186i_TENDENCIA_10+0.175i_TENDENCIA_20...
0.118	24	0.597i_HORA_PUNTA+0.339i_DIA_SEMANA_6-0.286i_INTERVALO_4_3+0.252i_DIA_SEMANA_5-0.196i_INTERVALO_8_1...
0.1132	25	-0.218i_FLAG_28+0.209i_FLAG_9+0.191i_FLAG_3+0.191i_FLAG_4-0.188i_FLAG_33...
0.1088	26	0.264i_FLAG_38+0.217i_TENDENCIA_45-0.214i_FLAG_45-0.209i_TENDENCIA_38+0.204i_FLAG_7...
0.1045	27	0.296i_FLAG_27-0.224i_TENDENCIA_27+0.22 i_FLAG_30-0.205i_FLAG_37+0.203i_FLAG_26...
0.1002	28	0.275i_FLAG_20+0.224i_FLAG_13-0.219i_FLAG_33+0.201i_TENDENCIA_1+0.191i_TENDENCIA_2...
0.0961	29	0.249i_FLAG_26-0.239i_FLAG_35-0.214i_TENDENCIA_26-0.207i_FLAG_32+0.207i_FLAG_28...
0.0922	30	-0.258i_FLAG_41+0.234i_FLAG_37+0.23 i_FLAG_39-0.224i_TENDENCIA_44+0.221i_TENDENCIA_41...
0.0884	31	0.282i_FLAG_11+0.282i_FLAG_10-0.249i_FLAG_16+0.245i_FLAG_24+0.241i_FLAG_21...
0.0852	32	-0.458i_HORA_PUNTA+0.349i_DIA_SEMANA_6+0.228i_DIA_SEMANA_5-0.203i_FLAG_9-0.18i_DIA_SEMANA_0...
0.0822	33	0.274i_FLAG_36-0.266i_FLAG_45+0.239i_FLAG_4-0.234i_FLAG_15+0.228i_FLAG_35...
0.0792	34	0.276i_HORA_PUNTA-0.271i_FLAG_30+0.243i_FLAG_42+0.219i_FLAG_38-0.218i_FLAG_29...
0.0763	35	-0.337i_HORA_PUNTA-0.325i_FLAG_45-0.299i_FLAG_12-0.262i_FLAG_3-0.257i_FLAG_29...
0.0733	36	-0.314i_FLAG_14-0.312i_FLAG_39-0.276i_FLAG_21-0.265i_FLAG_40-0.247i_FLAG_38...
0.0705	37	0.328i_FLAG_41-0.301i_FLAG_7-0.278i_FLAG_45+0.258i_FLAG_42+0.245i_FLAG_1...
0.0676	38	0.433i_FLAG_1-0.36i_FLAG_46+0.312i_FLAG_2-0.29i_FLAG_8+0.261i_FLAG_44...
0.0648	39	0.515i_FLAG_46-0.303i_FLAG_44+0.291i_FLAG_1+0.259i_FLAG_17+0.244i_FLAG_24...
0.0622	40	0.568i_FLAG_2-0.503i_FLAG_1-0.277i_FLAG_19+0.257i_FLAG_12+0.199i_FLAG_25...
0.0597	41	-0.393i_FLAG_26+0.364i_FLAG_45-0.348i_FLAG_19+0.293i_FLAG_37-0.261i_FLAG_2...
0.0572	42	0.584i_FLAG_32-0.366i_FLAG_41-0.344i_FLAG_44+0.218i_FLAG_43+0.206i_FLAG_40...
0.0548	43	-0.583i_FLAG_37-0.404i_FLAG_43-0.324i_FLAG_19+0.281i_FLAG_38+0.259i_FLAG_39...
0.0524	44	-0.508i_FLAG_26-0.44i_FLAG_12+0.314i_FLAG_20-0.237i_FLAG_1+0.227i_FLAG_2...
0.05	45	0.467i_FLAG_32+0.432i_FLAG_41+0.414i_FLAG_44-0.289i_FLAG_42+0.216i_FLAG_46...

Table 5: Selected attributes.

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34 ,35,36,37,38,39,40,41,42,43,44,45 → 45 attributes

Among all variables, the volume of incoming calls in previous intervals, night shift timetable, week of the month, time, intervals of hours (broken down in *2*, *4* or *8* hours) and intervals of peak hours must be highlighted and analysed for separate.

Figure 28 shows the behaviour when considering the previous time intervals. For almost all CGs, the optimum number of previous intervals required is usually around *5-6* intervals. Considering more previous intervals does not enable us to obtain better results and makes the learning process slower.

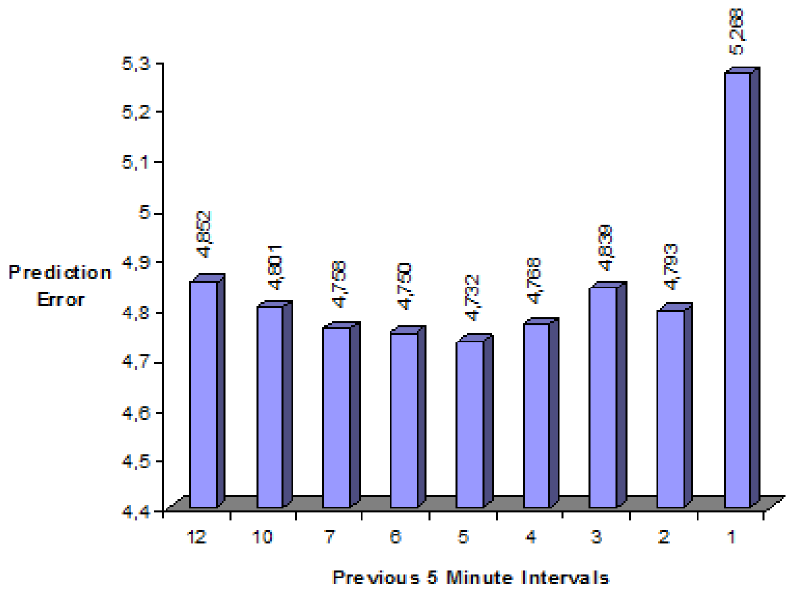


Figure 28: Mean absolute error returned by the ANN when considering previous 5-minute intervals.

Figure 29 illustrates a comparative study of the most relevant variables that have been studied in terms of error caused. The night shift timetable offers an upgrading of the results for every CG. When splitting days up into intervals of hours, predictions are also improved. The improvement coming from adding these hourly intervals might guide us to a wrong decision because these variables are correlated with the current number of incoming calls (our target) but the causality comes from the night shift timetable and peak time variables (it is more valuable to know peak hours rather than have hourly information). Note that the correlation among variables does not necessarily imply causality. In other words, the improvement is just obtained because these variables are correlated but only peak time intervals and night shift are truly useful to forecast the current number of incoming calls. Of course, PCA moderates the impact of these deceptive correlations.

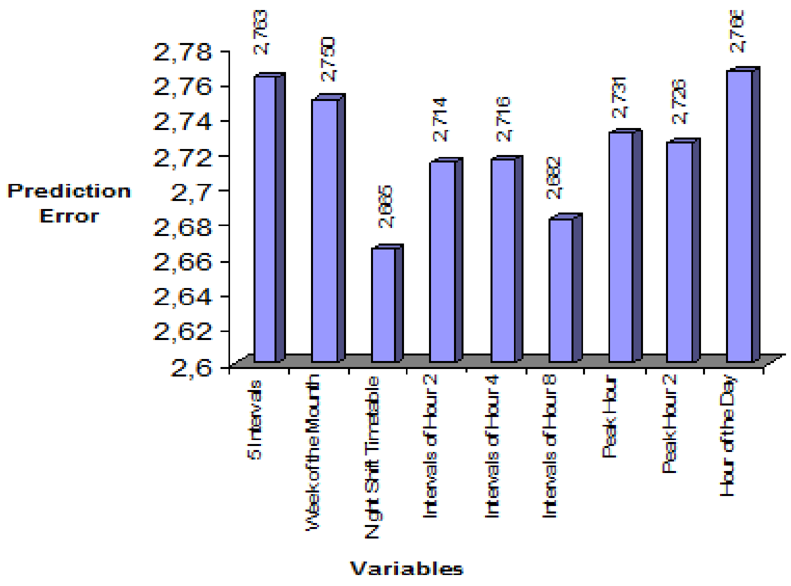


Figure 29: Mean absolute error returned by the ANN when adding different variables.

Intervals of peak hours are interesting to take into account because these divisions clearly outperform the results for almost all CGs. However, the upgrading is intuitively a bit lower for those CGs without many oscillations as there are fewer differences among day partitions.

The rest of variables influence the results in some CGs (like the week of the month) but not significantly enough for most of them (they slightly contribute to the target prediction). Since a quick response time is required and selecting more features involves a higher dimensionality, these variables have not been expressly included in our final implementation.

As Table 4 and Table 5 are difficult to follow, Table 6 summarises the most relevant individual features extracted from our dataset.

Table 6: Summary of the most relevant individual variables.

Individual relevant variables
Calls in Previous 0-5 Minutes
Calls in Previous 5-10 Minutes
Night Shift Timetable
Week day
Calls in Previous 10-15 Minutes
Calls in Previous 15-20 Minutes

Calls in Previous 20-25 Minutes

Minutes of the Day

Peak Time

Second Peak Time

5.4.3 Hidden Layer

We have already revealed that our ANN has a single hidden layer. Now, we have to determine the number of existing hidden units (hidden neurons) located in this layer. There are some (nonsense) rules to determine the number of neurons of the hidden layer(s) but, in our case, this number has been empirically determined.

Figure 30 shows the results obtained for a variable number of hidden units. We can appreciate that the optimum value seems to be around *20* hidden units as the mean absolute error gets minimised.

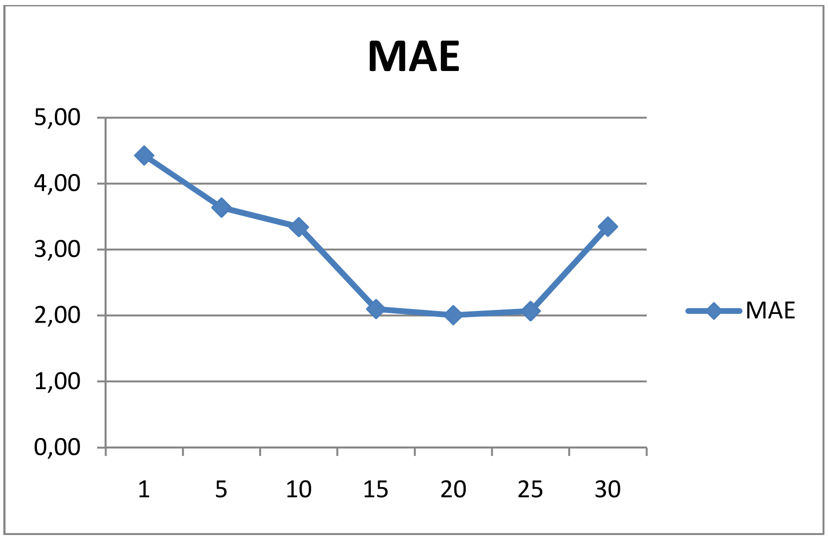


Figure 30: Mean absolute error, depending on the number of hidden units.

5.4.4 Output Layer

The last layer is named output layer and is used for unveiling the result of the prediction. The number of neurons of this layer is determined by the output variables. In our case, only one variable is predicted (number of incoming calls, available agents or abandonments) so that only one neuron forms the output layer. The output will be a floating number which indicates the number of incoming calls of a given CG, the number of available agents for a given CG or the number of abandonments for a certain CG (we need as many ANNs as CGs we have).

5.4.5 Parameter Initialisation

To initialise our ANN, we assign the predefined values stated in Section 4.4, except for the initial weights which must be randomly initialised. In particular, we set them up to small values ranged in *[-0.5, 0.5]*. The idea of initialising the weights in this way is to reduce the number of epochs during the training process. Starting from weights that are closer to the required ones will perceptibly necessitate fewer changes than weights that greatly differ.

Additionally, we can still outperform the results by including some specific knowledge of each CG. The large number of CGs (*1035*) and their miscellaneous behaviour make necessary to appropriately determine the initial parameters of the models (ANNs for each CG). To fulfil this requirement, the CGs have been divided into sets according to the mean number of incoming calls per day. This criterion has been taken as a consequence of the behaviour similarities of those CGs having similar volume of incoming calls. Therefore, we need to define different initial configurations for the step-sizes for these sets as well as the lower and upper bounds of the uRprop proposed in Section 4.3.3.

Table 7 demonstrates that we can still outperform the results a little by starting from different initial parameters depending on the CG behaviour. This table summarises the mean absolute error (MAE) gotten for *5* different CGs after *50* executions of their specific ANN.

Table 7: MAE obtained for *5* different CGs with/without sets for *50* executions.

Call Group	*MAE With Sets*	*MAE Without Sets*
CG1	2.84524	2.87196
CG2	2.34671	2.39941
CG3	4.32158	4.44656
CG4	1.40664	1.41888
CG5	0.83214	0.94906

The change proposed above does not vary our computing times but (slightly) improves the results, especially in those CGs that have more fluctuations in the arriving load.

5.4.6 Stopping Criterion

We consider the following measures to decide when to stop the training process:

1) *Maximum epochs reached*: the ANN will stop once a set number of epochs have elapsed (*1200*).
2) *Generalisation set mean squared error (MSE)*: this is the average of the sum of the squared errors (real – predicted) for each pattern in the generalisation set (*MSE < 1 incoming call*).

5.5 SEARCH MODULE ADAPTATIONS

This section comments the final configuration of the search module. We will describe the configuration of the evolutionary operators but, firstly, we will detail other important aspects.

5.5.1 Initialisation, Encoding and Population

We will encode the solutions as described in Section 4.4. Therefore, the solution consists in an array of integers whose indexes represent the available agents at a given instant and the array contents refer to the profile assigned to each agent. Figure 31 shows a fictitious example (related to Figure 21) of encoding for *9* customer calls (c_1-c_9) queued in *4* different CGs (cg_1-cg_4) depending on the nature of the calls, *5* agents (a_1-a_5) and *7* profiles (P_1-P_7), where $P_1=\{s_1\}$, $P_2=\{s_1, s_2\}$, $P_3=\{s_2\}$, $P_4=\{s_2, s_3\}$, $P_5=\{s_1, s_3\}$, $P_6=\{s_3\}$ and $P_7=\{s_4\}$. Now, suppose that the agents have the following potential skill profiles: a_1~$\{P_1,P_2\}$, a_2~$\{P_1,P_3,P_7\}$, a_3~$\{P_4,P_5\}$, a_4~$\{P_6\}$ and a_5~$\{P_2,P_3,P_7\}$. We have seen the potential profiles for every agent but only one profile can be assigned to each agent at a given instant *t*; therefore, a feasible solution would be Figure 31. Note that more than one agent can have assigned the same profile (e.g. a_1 and a_5).

Index (agents) →	1	2	3	4	5
Content (profiles) →	2	7	4	6	2

Figure 31: Example of encoding for an MSCC.

The population contains *20* different individuals encoded as hinted above. In our case, we propose to start from a randomly generated initial population, including the best solution found in the previous time-frame because the configuration of agents' profiles should not change too much over two successive time-frames (consecutive states).

5.5.2 Fitness Function

Now, we present the fitness function which is defined over the proposed encoding to measure the quality of a given solution. Our fitness function is inspired in the estimation of the *total service level* provided in [12] although we also consider the priority of each CG weighted as follows:

Total_service_level = $\sum_{i=0}^{k}(Pr_i \times SL_i(\gamma_i,\alpha_i))\times\mu \quad \{sl:\Re\times[0,1]\times[0,1]\rightarrow[0,1]\}$ where *k* refers to the number of CGs, μ is a normalising factor $(1/\sum_{i=0}^{k}Pr_i)$, *Pr*$_i$ is the priority of the CG$_i$ whose service level is defined as $SL_i(\gamma_i, m_i) = 1 - P(Agents\_are\_busy)\times e^{-(\gamma_i - m_i)\frac{\tau_i}{\beta}}$ given that $P(Agents\_a\ re\_busy) = \left[1+\frac{\gamma_i\ \text{-}\ m_i}{m_i}\sum_{\zeta=0}^{\gamma_i-1}\frac{(\gamma_i-1)...(\zeta+1)}{m_i^{\ \gamma_i-\zeta-1}}\right]^{-1}$ where γ_i is the load of **(16)**
CG$_i$ (number of incoming calls of CG$_i$ by the mean processing time: $n_i\times\tau_i$), *m*$_i$ is the number of agents of CG$_i$ (based on the profiles assigned in the chromosome), τ_i is the number of agents of CG$_i$ and β is the duration of the time-frame expressed in seconds.

Additionally, we handle some hard and soft constraints derived from the business rules given by our business units. In our case, these constraints are associated to tasks, agents, timing, actions or desired/undesired scenarios. Thus, the algorithm cannot violate hard constraints (e.g. we cannot change agents' profiles continuously due to certain laws and regulations); although we allow certain movements which may imply the violation of some soft constraints (e.g. we should not take agents from CGs in which the service level is below a given threshold). Undoubtedly, this type of movements is penalised according to the degree of non-accomplishment of these constraints and their relevance as described in Chapter 2. Therefore, the *fitness function* can be formalised as follows:

$$f\ = (total\_ser\ vice\_level\ \text{-}\ penalisati\ ons\_constr\ aints)\quad f:[0,1]\times[0,1]\rightarrow[-1,1] \qquad \textbf{(17)}$$

where *penalisation_constraints* is the value obtained after applying our business rules (e.g. agents from CG-*i* should not move to CG-*j*).

Finally, we can speed-up the evaluations by introducing a *partial fitness function*. The first time, we need to employ (17) but the rest of the time; we just need to evaluate those groups affected by a mutation or, in the case of the LS, when generating a new neighbour. Hence, we only process the affected CGs in (16) and update their original values. With this information, we then recalculate (17).

5.5.3 Evolutionary Operators

In this section, we explain the final configuration of the evolutionary operators described in Section 4.4. This configuration is the following one:

*Selection***:** Since the population needs to be bred each successive generation, we have chosen a binary tournament selection.

*Crossover***:** The following step is to produce a new generation from selected individuals. We consider that children will inherit the common points in their parents (potentially, the best genes) and randomly receive the rest of genes from them.

*Mutation***:** This operator causes tiny changes in the genes of the chromosome to explicitly maintain diversity (actually there are much more mechanisms). We apply a perturbation over each gene of the chromosome with a probability of *0.03*. This perturbation corresponds to changes of profiles in some agents (e.g. agent a_2 who had assigned the *profile* P_1 has now associated the *profile* P_3 due to a mutation).

Replacement policy: Finally, we decide which individuals are incorporated (or maybe reinserted) into the population. In this study, we consider elitism with a probability of *0.93* to replace the worst individuals of the population for next generation. And, with a probability of *0.07*, a worse individual may be captured. Note that our MA relies on a steady-state scheme.

The configuration proposed above has not been chosen ad-hoc. Instead, we have evaluated different configurations and selected the best one.

Figure 32 shows the most relevant configurations that we have tested out during *600* seconds (*10* minutes). *Y*-axis represents the fitness value while the *X*-axis stands for the number of generations. Note that we do not apply the LS mechanism over the individuals at this stage.

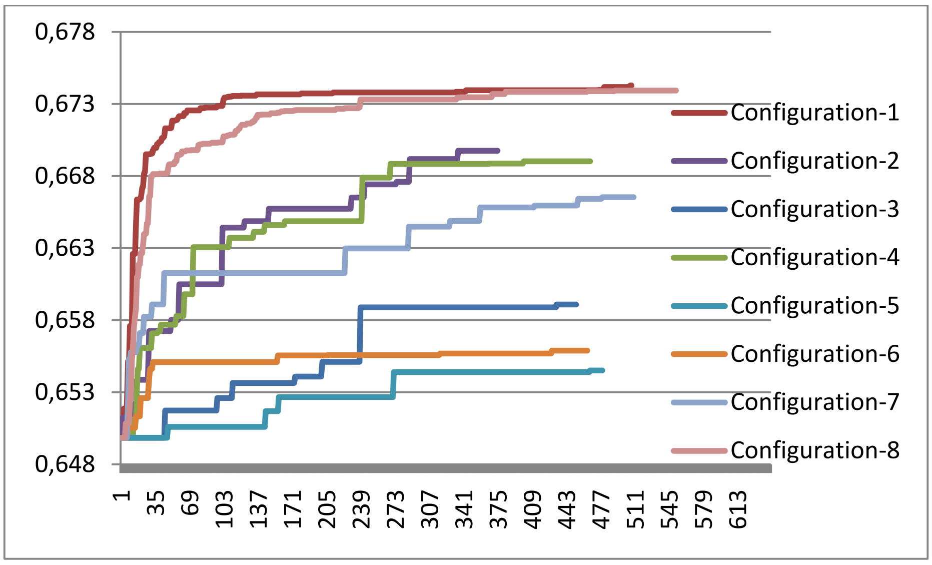


Figure 32: Fitness obtained for *8* different configurations of evolutionary operators.

Configuration-1 refers to the configuration described above. We perceive that Configuration-1 can process many generations compared to the rest of configurations (excluding Configuration-8). Configuration-1 also obtains the best fitted solution. The convergence is favourable for a 5-minute execution (around generation-300). Note that most time-intervals have that duration so that the improvement during a complete day is noticeable. After that point the improvement is minor although we can observe another important slope around generation-500. When the dynamism is high, this configuration is also very appealing because this configuration steeply slopes. Besides, when the time-frame increases, the configuration is also appropriate as it still goes on improving the fitness value.

Configuration-2 differs from Configuration-1 in the mating-selection as it considers mating by similarity. For this reason, the number of generations is reduced. This configuration allows for diversity but the convergence is slower than Configuration-1. Instead, Configuration-2 almost always improves and may be good for stable systems (longer time-frame).

Configuration-3 applies a mating based on the differences on the fitness values. Each individual is mated with its most different individual in terms of fitness: highest difference in fitness value. We notice that this mating operator is faster but the results seem to be worse than Configuration-2.

Configuration-4 applies a random mating and a special selection in which the best fitted individual is taken as well as its most different individual in terms of genes (like similarity but for final selection). This configuration always increases and has a nice slope although the two first configurations seem to better behave.

Configuration-5 imitates Configuration-4 except for the mating mechanism. In this case, each individual is mated with its most different individual in terms of fitness. It has a poorer performance and can carry out fewer generations.

Configuration-6 employs a mating by highest fitness difference and applies a replacement policy in which the best fitted individual and the worst fitted individual after reproduction fully replace the best and the worst individuals from the populations respectively. The performance and slopes are poor.

Configuration-7 proposes a similar scheme to Configuration-6 except for the random mating. More generations are carried out and better performance than Configuration-6.

Configuration-8 applies a random mating and our crossover inspired in simulated annealing. We also consider elitism for the replacement policy and mutations by ranges. We also perceive a fine slope at the beginning. If we had more time, we might increase the initial temperature value to explore more at the beginning. Potentially, we could obtain better fitted individuals.

5.5.4 Memetic Algorithm

Once we have the evolutionary operators, we need to define the refinement mechanism and select a target subpopulation to refine. Another important issue is the refinement frequency.

Refinement algorithm: The refinement mechanism is a basic LS based on the best neighbour scheme as described in 4.4.3.1.

Subpopulation for LS: The LS is applied over the best *25%* of individuals.

LS frequency: The LS is applied over the selected individuals each *10* generations.

5.5.5 Parallel Memetic Algorithm

The last step is to parallelise the MA. We will apply an island model with several subordinate islands connected to a master island as follows:

Topology: We consider a star topology with *4* subordinate islands (as Figure 33 illustrates) which correspond to "simple" MAs. These islands are connected to a master island (another "simple" MA which coordinates and synchronises the rest of islands).

Migration: Each subordinate island sends the *10%* of the best fitted individuals when the master island asynchronously demands these individuals to the rest of islands.

Replacement policy: We will apply elitism so that the best fitted individuals from the subordinate islands will replace the less fitted individuals from the master island's population whether these individuals are better fitted.

Migration frequency: Each *50* generations, the master island blocks the rest of islands to ask them for their best fitted individuals.

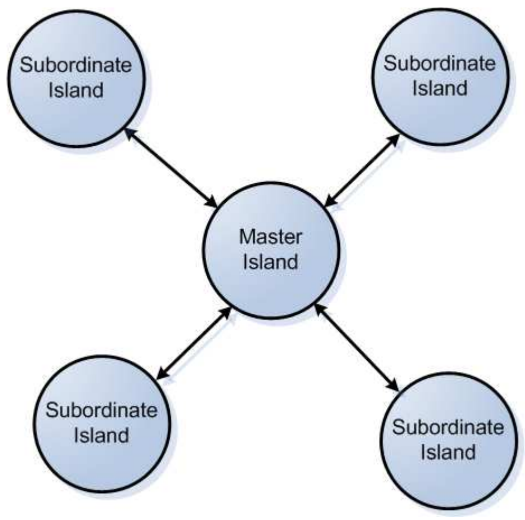


Figure 33: Star topology with *4* subordinate islands and a master island.

CHAPTER 6. EVALUATION OF THE APPLICATION

This chapter evaluates our approach on the MSCC domain. Section 6.1 describes the dataset employed. Section 6.2 points out the hardware descriptions of the SunFire sever in which the evaluations have been performed. Section 6.3 analyses the selected metrics for testing and comparing our approach. Section 6.4 examines the forecast module for five different CGs as there are too many to accomplish an exhaustive study for all of them. Section 6.5 evaluates the search module by studying several time intervals from days with different complexity; this section also compares our search module with other acknowledged techniques. In Section 6.6, we will analyse our complete approach (forecast module + search module) for one-day campaign. We will also compare how our complete approach outperforms other conventional call centre's algorithms.

6.1 DATASET EMPLOYED

An important step consists in creating a suitable dataset, hunting for a fair balance between the amount of data and a representative period of time measured in terms of days, carved up in hours and minutes (microscopic level → fine grain). In our case, we will work with *45* numeric attributes (see Section 5.4) and thousands of registers which correspond to *5*-minute interval information from records stored during several months.

Besides, the number of selected days must be a multiple of *7* because the predictor *week-day* has imperative influence on the training and validation processes as Section 5.6 demonstrates. Moreover, the number of days must be large enough to represent every possible pattern (cases). Therefore, the number of days to take into account should be, at least, *91* days in order to cover all possible patterns with the aforesaid considerations. It is very important to divide data like this as this composition allows for trend and seasonality detection.

Our problem presents *1035* CGs; hence, the dataset is too large to do an exhaustive study for all of them (obviously, the forecast module has been trained, considering every CG). Consequently, *5* representative CGs with different behaviour in terms of oscillations, arrival rates, processing times and nature, have been carefully picked in order to perform a generic enough approach (see Figure 34). These oscillations intuitively imply a higher complexity. We have assigned an index to each CG that designates their complexity level which ranges from the most convoluted CG to the simplest one (labelled from *1* to *5*). For business reasons, we are not allowed to reveal real CGs names but this is something we should not be concerned about in this work.

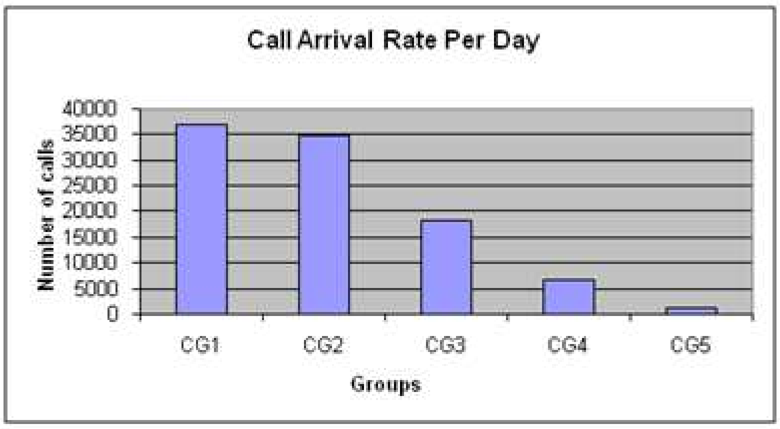


Figure 34: Call arrival rate per day, grouped by CGs.

Afterwards, a different model has been exclusively developed for each CG because of differences among CGs. Then, the whole dataset has been split into subsets, contemplating every CG.

Once we have a single dataset for each CG, this is shuffled and then randomly divided into three subsets, following the cross-validation structure [115] (see Figure 35): Training (55%), generalisation (20%) and validation (25%).

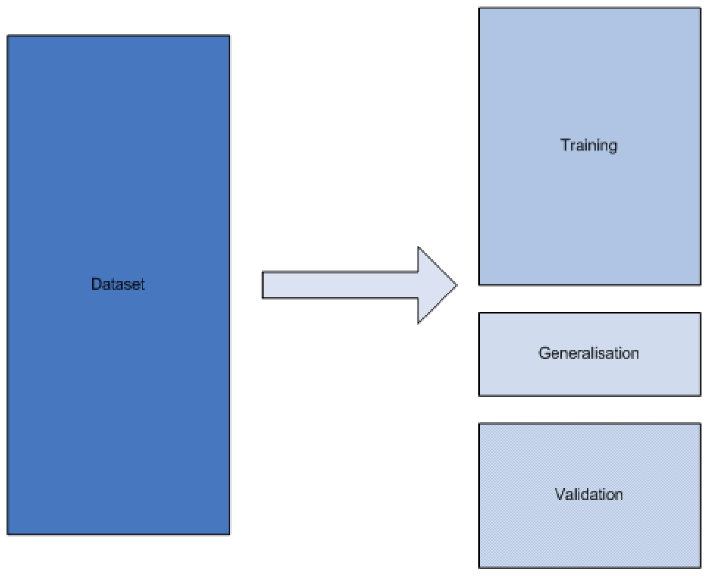


Figure 35: Dataset partitions: training, generalisation and validation.

The training dataset, which is the largest partition, is used for training our ANN. Instead, the generalisation partition is used at the end of each epoch to observe whether our ANN correctly handles unseen data. Once the training process has finished, the validation partition is showed to our ANN to determine its real precision.

Although we will analyse *5* CGs for the forecast module in Section 6.4, we must validate the search module by considering all the CGs. To evaluate the search module, we have chosen several time intervals from days with different complexity (see Section 6.5). Therefore, we can discover the benefits of our approach, depending on the dynamism of the system. Besides, we will analyse our complete approach (forecast module + search module) for one-day campaign in Section 6.6.

6.2 HARDWARE DESCRIPTION

This section describes the key hardware features of our *SunFire 4900* server in which all experiments have been launched. These features are the following ones:

1) *64*-bit Chip Multithreading UltraSPARC® IV technology, with over *2x* the throughput of previous generations.
2) Scales up to *8* x *1.35*-GHz UltraSPARC IV CPUs with *16* MB L2 cache per processor.
3) Up to *16* simultaneous compute threads with up to *64* GB memory.
4) Solaris-TM 8, Solaris 9, and Solaris 10 Operating System.
5) Robust capabilities in the Solaris 10 OS such as predictive self-healing to increase reliability, Solaris containers for increased utilisation, and dTrace to optimise application performance.
6) Industry leading price/performance and benchmarks.
7) *9.6* GB/second SunTM Fireplane interconnect.
8) N+1 hot-swap power supplies/hot-pluggable disks.
9) Sun systems controller for remote system administration.
10) Automatic system recovery to maximise uptime.
11) Integrated fibre channel disk subsystem, multi-pathing-ready, supporting up to 12 FC-AL disks.
12) 9 PCI slots help ensure a highly scalable, well-balanced system.
13) 17-RU tower/desk-side, rack mountable.

6.3 METRICS

Metrics are usually specific for a given subject area and are often valid only within a certain domain so that these cannot be directly interpreted outside it. We have selected several metrics to evaluate our forecast and search modules as well as the complete process.

6.3.1 Forecast Metrics

In order to make the forecast process more understandable, we define the error, *e*, as the difference between the real output value, *f*, and the predicted output, y.

To evaluate the forecast module, we will apply the following metrics:

1) *Mean absolute error* (MAE): average of the absolute errors: $MAE = \frac{1}{n}\sum_{i=1}^{n}|f_i - y_i|$.
2) *Standard deviation* (SD)*:* the standard deviation of a statistical population is the square root of its variance.

6.3.2 Search Metrics

In order to compare all the search algorithms in terms of quality of the solution, a metric to represent that quality is required. We presume that solution quality comparisons must be made over the same problem instances. Comparisons over different problem instances are normally weaker as those instances may have dissimilar structures so that the conclusions might be completely erroneous.

To evaluate the search module, we will examine the following metrics:

1) *Worst solution*: Less fitted value, considering *e* executions (*50*).
2) *Best solution*: Best fitted value, considering *e* executions (*50*).
3) *Mean solution*: Mean value from *e* executions (*50*).
4) *Standard deviation*: Standard deviation from *e* executions (*50*).
5) *Performance*: Ratio of the current fitness value with respect to the best fitted value. It can be calculated as follows: $Performance = \frac{current\ fitness\ value\ (technique_i)}{best\ fitted\ value} \times 100.$

6.3.3 Call Centre Metrics

Most MSCCs employ more than *30* different metrics to verify how operations are going. However, sometimes, just observing a subset of variables may accomplish our goals. Metrics refer to customer satisfaction, quality, productivity, agent utilisation or costs per contact (for outbound MSCC).

To evaluate the complete approach, we will consider the most important metric, from our point of view, for any MSCC: the *mean service level* which is defined as the percentage of customer calls that have to queue shorter than a specified amount of time (in our MSCC, *20* seconds). This metric covers aspects such as quality, productivity, client satisfaction among others.

6.4 COMPARISON OF FORECAST TECHNIQUES

Now, we come to an important point that cannot be overlooked. We need to compare our forecast module with other techniques in order to verify whether (or not) our approach is convenient for a real-world DMAS.

In Section 4.3.1, we have reviewed the previous work on forecast techniques. Now, we proceed by comparing our approach based on ANNs (uRprop) with the most relevant forecast methods from the state-of-the-art. As we cannot implement all the techniques presented in Section 4.3.1 for this work, we have chosen R's forecast package [116] to evaluate them. Note that we have selected R forecast package because it is very well-implemented and is an open-source so that we can analyse the inner of the implementation. Other data mining tools such as SPSS or SAS have also truly potent algorithms, but we should not apply them for this comparison as we would not have any insight about the algorithm behind them.

To compare the techniques described in Section 4.3.1, we have selected five representative CGs as described in Section 6.1. Our dataset is composed by thousands of records extracted from our production environment during *91* different days. We have trained the models with this dataset and validated them with the following two weeks. This means that the validation has been carried out by means of continuous (online) predictions for a *2*-week time period. Although this validation should be convincing, we have executed the models *49* more times offline (a total of *50* executions). The models have been run under two of the cores of our SunFire 4900 server (Solaris 10).

Now, let's go to the thick of the comparison. Figure 36 and Table 8 illustrate the MAE comparison between time series, ARIMA, linear regression, logistic regression and our ANN. This confirms that, although each CG has a different behaviour and needs a different model to obtain the best approximation, our ANN regularly behaves better than the other techniques in our domain (we do not claim a universally better approach). While ARIMA and time series emphasise the "recent past", our ANN is more flexible because it not only considers previous tendencies and time points but also covers historical patterns from other days and other contextual information (e.g. if there is a commercial campaign, the dynamism will be higher). The capability of considering historical data is really valuable as we can discover interesting features like the peak hours' effect and more complex relationships.

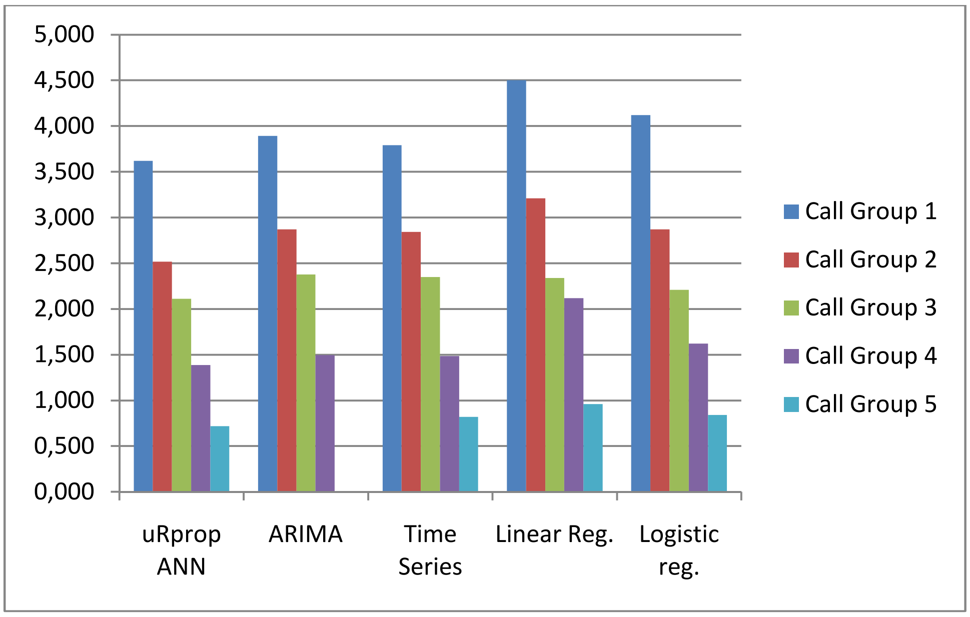


Figure 36: General Comparison (MAE is in the Y-axis and techniques are in the X-axis).

Looking at Table 8, we also perceive that our ANN has less variability in the predictions as the standard deviation is usually lower, except for the call group-1 (time series) and the call group-4 (logistic regression) where our standard deviation is slightly higher.

Another remarkable result is that time series and ARIMA outperform regression models in most CGs, probably, due to the capability of considering trend and seasonality rather than simply considering relationships between the dependent variable (output) and any one of the independent variables when these vary.

On the one hand, the main problem with our approach is that we need longer training times than ARIMA, time series or regression models. Besides, our implementation is much more complex than these techniques. When not having a multimodal space, we recommend simpler techniques: linear regression for linear dependencies and times series for stable patterns or those that merely depend upon the recent past.

On the other hand, although there are no huge differences between our approach and the rest of the abovementioned techniques in terms of MAE for a given instant, we strongly recommend our approach as far as slight differences in terms of MAE for a given point may induce huge mean errors throughout a day (cascade effect).

Table 8: MAE & SD Comparison.

Call Group	*uRprop*		*ARIMA*		*Time Series*		*Linear Regression*		*Logistic reg.*	
	MAE	*SD*	*MAE*	*SD*	*MAE*	*SD*	*MAE*	*SD*	*MAE*	*SD*
Call Group 1	3.621	6.381	3.892	6.430	3.791	6.360	4.503	6.833	4.120	6.501
Call Group 2	2.519	4.472	2.872	4.851	2.841	4.821	3.210	5.756	2.870	4.507
Call Group 3	2.112	3.592	2.378	4.087	2.349	4.104	2.340	3.843	2.210	3.551
Call Group 4	1.387	2.752	1.494	2.785	1.486	2.771	2.120	3.154	1.620	2.718
Call Group 5	0.718	1.405	0.823	1.408	0.819	1.406	0.960	1.667	0.840	1.313

Up till now, we have demonstrated that our approach outperforms several forecast techniques but, can we beat other ANN's learning algorithms? Our approach often gets trapped in local minima so that we cannot claim that our approach can behave better than other learning algorithms for ANNs as there are other nice exact approximations. However, our environment is very dynamic and we have a very limited time to train the models (we need a model for each CG). So, can we outperform other learning algorithms given our timing constraint? Fortunately, the answer is "yes". At least, we can offer more accurate results for those CGs which are very dynamic and have more incoming calls. This makes sense because the search space is more complex and the exact approximations have no time to compute the complete process. Nonetheless, other learning algorithms might behave better for those CGs with fewer calls (e.g. CG4 or CG5). Obviously, this type of CGs is not really relevant as only few calls arrive.

Figure 37 demonstrates that our learning algorithm outperforms Backpropagation, Quickpropagation, classical Rprop, Rprop with weights backtracking, ANNs with pruning and also ANNs with exhaustive pruning for the most convoluted CGs (CG1 and CG2). For the easiest ones (CG4 and CG5), learning with exhaustive pruning and Rprop with weights backtracking outperform our uRprop. If we let other learning algorithms run during more than *1200* epochs, the differences would be higher. However, we have a very limited time to train the models and our approach behaves better than the other techniques for the most complicated CGs.

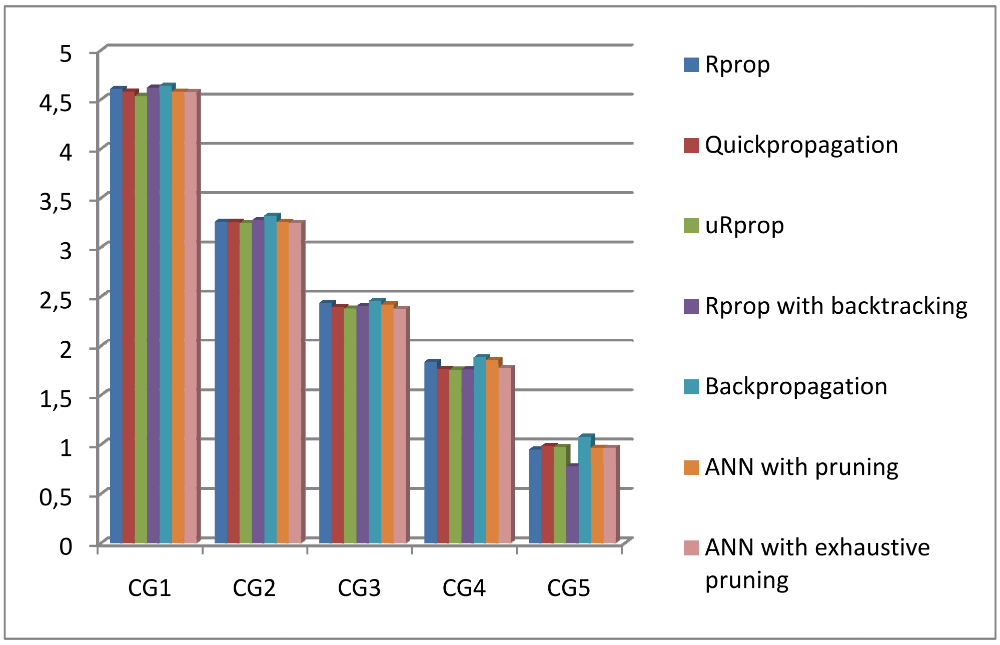


Figure 37: NN Comparison (SSE x epochs).

The main disadvantage with gradient-descent methods is premature convergence to local optima. Occasionally, local optima can be likewise nearly global optima although these normally hurt performance. This difficulty can be overcome by using global optimisation methods although these techniques are highly time-consuming and become less appropriate for dynamic systems. Learning with exhaustive pruning and Rprop with weights backtracking obtain better results than uRprop when there are no timing constraints or the search space is simple enough to find the optimum in few epochs.

To conclude, we can state that demanding dynamic systems need fast (re)trains as long delays may drastically damage performance, even when reached solutions deem optimal.

6.5 Comparison of Metaheuristics

In the previous section, we have compared our forecast module with other acknowledged predictive techniques. We have seen that our approach is not universally the best one as other learning algorithms outperform our uRprop when these have more time to train the models (the ANN for each CG) or when the dynamism is low. Fortunately, our learning algorithm behaves better for those CGs with higher dynamism when the time to train the models is reduced. But, we still have to determine the right assignment among agents and tasks, given the predictions provided by the forecast module. This task is carried out by our search module which will be compared with other famous MHs.

Now, similarly to the previous section, we describe the (two) problem instances (medium and high difficulty, respectively) that we have created from our dataset to test out our search module. For a fair comparison, every MH will be run over the same problem instances *50* times. These two problem instances are composed by real data taken from our MSCC's production environment during two different days at the same hour (from *12:40* to *12:45*, *300* seconds): a one-day campaign and a normal day. The size of the time-frame to execute all the MHs has been *300* seconds (*5* minutes) because it is the commonest time-frame size. We have selected the interval *[12:40-12:45]* as it is precisely the most critical hour of the day (highest load of the day: *n/m*). Note that around *800* incoming calls (*n*) simultaneously arrive during a normal day in such a time interval, whereas up to *2450* simultaneous incoming calls may arrive during this interval throughout a commercial campaign. The number of agents (*m*), for each time interval, oscillates between *700* and *2100*, having *16* different skills for each agent on average (*minimum=1* and *maximum=108*), grouped in profiles of *7* skills on average. The total number of CGs considered for this study is *167*. Therefore, when the workload (*n/m*) is really high, finding the right assignment among agents and incoming calls becomes fundamental. In this way, we have run every MH under two double-core processors of a *Sun Fire E4900* server (one processor for the interfaces and data pre-processing, and the other one for each MH).

Once the magnitude of our problem instances has been presented, each MH is compared alongside the others. Table 9 summarises the results obtained by each MH in *50* executions, starting from *50* different randomly generated initial solutions.

In our comparative study, we present dissimilar MHs which cover diverse search strategies. Theoretically, due to the local character of the basic LS, it is complicated to reach a high-quality solution because the algorithm usually gets trapped in a neighbourhood when a local minimum is found. This occurs because the engine is always looking for better solutions which probably do not actually exist in the neighbourhood. For this reason, sometimes, it is more appropriate to allow deterioration movements in order to switch to other regions of the search space. This is precisely the shrewd policy of SA whose temperature allows for many oscillations (the probability of accepting a worse solution decreases according to the time) at the beginning of the process and only few ones at the end (fewer chances to select a worse solution as the algorithm is supposed to be refining the solution at this point). Specifically, we have chosen Cauchy's criterion because the convergence is faster than Boltzmann's and we only have *300* seconds to run the complete process. In Cauchy's scheme, the temperature is defined as $T_{it} = T_0 / (1 + it)$, where *it* is the iteration number and the initial temperature is $T_0 = (\mu / -\log(\Phi)) * f(S^*)$ where *f(S*)* is the cost of the initial solution, *Φ* stands for the probability of accepting a "μ" worse solution than the current one ($\Phi = \mu = 0.3$). Besides, this scheme avoids decreasing the distance between two solutions when the process converges (jumps in the neighbourhood). Therefore, the temperature must be high enough at the beginning to better explore the search space (its neighbourhood) and low enough at the end to intensify the search as well (exploitation of promising areas). The stopping condition must agree with the number of neighbours generated. The maximum of neighbour solutions generated each time is *L(T)=30* and the probability of accepting a worse solution is $exp(-\delta/T_{it})$ given that *δ=f(Neighbour_Solution)-f(Current_Solution)* and T_{it} is the temperature at iteration *it*.

We perceive from Table 9 that SA behaves worse than the other MHs except for the easiest instance of the problem. This may occur because we are not plenty of time in our environment and the power of SA relies on a progressive cooling. If we cool off the temperature too fast, we are missing the effectiveness of accepting worse solutions in some cases. Instead, if we cool off the temperature too slowly, we may be accepting worse solutions systematically without converging. We have applied a trade-off between exploration and exploitation but the computing time (*300* seconds) seems to be limited to apply SA to our environment (perhaps, things might change when having more time).

Another option to increase the diversity in the solutions is to enlarge the environment, as VNS does. This philosophy consists in making a systematic change upon the environment when the LS is used, increasing the environment when the process gets stagnated. In the VNS, the search is not restricted to only one environment as in the basic LS; instead, the neighbourhood changes as the algorithm progresses. In our experiments, we have considered three different environments $e_{max}=3$: $(\mathrm{e}_1 \rightarrow \mathrm{nh}_1 = 0.3 \times n; \quad \mathrm{e}_2 \rightarrow \mathrm{nh}_2 = 0.5 \times n; \quad \mathrm{e}_3 \rightarrow \mathrm{nh}_3 = n)$. These steps are repeated during *300* seconds (our stopping condition). Albeit we only consider three distinct neighbourhoods, the improvement of the VNS compared to basic LS is noteworthy. Consequently, the remarkable factor becomes the change in the number of neighbourhoods and their sizes as well as considering how the algorithm reacts in response. Table 9 also shows how VNS only slightly outperforms SA for the hardest instance of the problem.

Another strategy is to start from different initial solutions as ILS accomplishes. ILS generates a random initial solution and afterwards applies a basic LS. Subsequently, this solution is systematically mutated and thus refined. For ILS, the complete process is repeated during *300* seconds wherein the LS is the one proposed in Section 3.1.1 and the perturbation affects to the *3%* of agents. We can observe that ILS obtains solutions which vaguely improve those given by SA and VNS for the hardest problem instance, although it performs worse for the simplest problem instance as Table 9 corroborates.

Another way to find an accurate solution involves using methods based on populations, such as MAs. If the diversity of the solution is low, then the MA converges to the closest neighbour. Nevertheless, when the selective pressure is high, individuals may be alike or even identical. To speed-up convergence, MAs apply an LS procedure upon a set of chromosomes (candidate solutions) that are refined every certain number of generations. Incorporating a hybridisation mechanism to the GA is valuable as the algorithm is improved in all respects (exploration and exploitation). The configuration of the GA's operators is the one provided in Section 5.5.3 whereas the LS mechanism is given in Section 3.1.1. Table 9 points out how our MA not only outperforms all the presented MHs for both problem instances but also remains more unwavering (less differences among best, worst and mean fitness values).

Table 9: Results obtained by the MHs in *50* executions starting from random initial solutions for two problem instances: medium and hard (larger number of incoming calls and high variability). Values refer to the fitness obtained by all the MHs.

Algorithm	*Best solution*		*Worst solution*		*Average*		*Standard dev.*		*Effectiveness*	
	Medium	*Hard*	*Medium*	*Hard*	*Medium*	*Medium*	*Hard*	*Medium*	*Hard*	*Medium*
MA	0.796	0.758	0.785	0.751	0.796	0.754	0.001	0.001	100	100
ILS	0.768	0.728	0.755	0.722	0.763	0.725	0.002	0.003	95.85	96.15
VNS	0.790	0.727	0.766	0.723	0.775	0.724	0.005	0.001	97.36	96.02
SA	0.782	0.721	0.773	0.709	0.779	0.716	0.001	0.003	97.86	94.96

It is important to remark that differences among techniques are not huge after reaching a fitness of *0.8* since the complexity exponentially increases in our environment. Therefore, minor improvements on the fitness value after that point are hard to obtain but very valuable to accomplish a fair workforce distribution.

Hitherto, we have demonstrated that our (single) MA has been able to outperform other MHs in our real-world production environment. However, we described our search module as a parallel MA based on an island model (star topology) with *4* subordinate islands connected to a master island. So, can this architecture obtain better results than the single MA? Certainly, yes it can. Nevertheless, the improvement, which is remarkable, cannot be impressive as the complexity increases asymptotically. Table 10 compares the results obtained by the parallel MA with those obtained by the single MA. The parallel MA improves the results of the single MA in a *4%* for the hardest problem instance and *6.8%* for the easiest one. Although there is no a linear increment of fitness, the results are definitely better. These also converge faster but we fixed the computing time (*300* s.).

Table 10: Results obtained by our single and parallel MAs in *50* executions starting from random initial solutions for the two problem instances studied. Values refer to the fitness.

Algorithm	*Best solution*		*Worst solution*		*Average*		*Standard dev.*		*Effectiveness*	
	Medium	*Hard*	*Medium*	*Hard*	*Medium*	*Medium*	*Hard*	*Medium*	*Hard*	*Medium*
PMA	0.834	0.818	0.823	0.783	0.829	0.809	0.003	0.002	100	100
MA	0.796	0.758	0.785	0.751	0.796	0.754	0.001	0.001	96.01	93.20

6.6 COMPARISON OF WORKLOAD DISTRIBUTION ALGORITHMS

We have analysed the two main modules of our approach (forecast module + search module) and seen that these outperform other famous techniques for separate. Now, we will combine these modules and introduce the dynamic time-frame described in Chapter 4. Specifically, we will compare our approach throughout a demanding working day (there was a commercial campaign during the day which has been measured). In this way, we have run the algorithm over a whole day with approximately *315.000* calls (up to *28.800* calls/hour and *2.450* simultaneous calls) under *12* double-core processors of a *Sun Fire E4900* server (one for the interfaces, another one data pre-processing, another one for the database, two processors for controlling, two processors for the forecast module, and the last five ones for the search module) with *96GB RAM*. The mean number of agents in each time-frame is *2.100*, having *16* different skills for each agent on average (*minimum=1* and *maximum=108*). The total number of CGs is *820*. The mean processing times differ a lot, depending on the CG (from seconds to minutes). All data were taken from our MSCC.

Now, we compare our approach with classical SBR [106], ED-SBR (an improvement of classic SBR [106]) and Koole's algorithm [11]. Figure 38 illustrates the real service level given by these techniques during a demanding working day. The graphs compile the real service levels for each CG, considering the relevance (weight) of each one. Since incoming traffic mainly arrives from *9* a.m. till *8* p.m.; therefore, we need more accurate results for this time-interval and, particularly, for the peaks which occur around 13 p.m. (see point *32* in Figure 38), *15* p.m. (see point *66* in Figure 38) and *19* p.m. (see point *100* in Figure 38) because, in these points, the load is much higher. Our approach clearly improves the results reached by other algorithms in these critical points (peaks). For the rest of points, we see that our algorithm usually better behaves than the rest of techniques. Classic SBR and ED-SBR sometimes offer a similar configuration of agents than our approach for some time points and, consequently, the same service levels; but, on average, the service levels are clearly worse than ours. Only in few points, the service level provided by ED-SBR and SBR is slightly higher than ours (e.g. around 11:45, point *17*). This happens because in these points, our predictions had a greater error and SBR and ED-SBR consider the current state of the system. However, we can see that differences are tiny in these critical points and we present more stable results over the time. This corroborates that an adaptive middle-term time-frame is recommended as algorithms can reach nearly optimal solutions while short-term algorithms often collapse in local optima. But, short-term algorithms present a high adaptability to changes that long-term time-frame techniques cannot cope with. These long-term based techniques generally extract patterns from the historical and are only appropriate for stable environments. For this reason, our algorithm and SBR outperform Koole's approach which is designed for more stable MSCCs. Koole's algorithm finds very accurate solutions when the dynamism is more reduced such as classical staffing. Nevertheless, this is not the case of our environment and this kind of techniques cannot be efficiently applied to our MSCC.

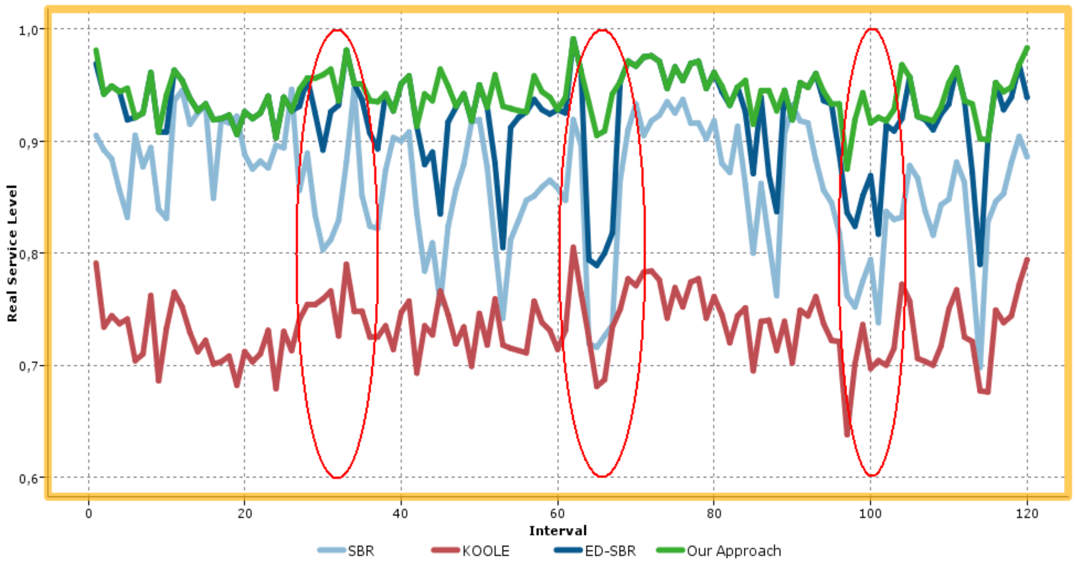


Figure 38: Service level given by different techniques for a whole campaign day. X-axis represents intervals of 300 seconds and Y-axis represents the real service level (not a fitness value).

Table 11 compares the results obtained by all techniques presented in Figure 38. Table 11 presents the mean service level for *120* intervals, its standard deviation and the effectiveness, considering that our method represents the highest performance. Note that we are actually comparing the behaviour of our approach with other conventional techniques during a complete day rather than focusing on specific time-frames as we have presented till now. It is crucial obtaining accurate results for isolated time-frames but we cannot obviate that we are executing our approach continuously so that the transitions among system states (for each time-frame) must be taken into account.

Table 11: Comparison of our approach with other relevant (call centre) algorithms for *120 5*-minute intervals.

Algorithm	*Real service level*	*Standard deviation*	*Effectiveness*
Our Approach	0.941	0.020	100
ED-SBR	0.901	0.043	95.757
SBR	0.860	0.056	91.405
KOOLE	0.733	0.029	77.896

CHAPTER 7. CONCLUSIONS AND FUTURE WORK

The present chapter summarises the ideas exposed in this dissertation and highlights the major contributions of our work in Section 7.1. We also give some guidelines for future work in Section 7.2.

7.1 SUMMARY AND CONCLUSIONS

We have presented a novel approach to the problem of workforce distribution in dynamic multi-agent systems based on blackboard architectures (common repository of knowledge). We have seen that these systems are extremely complex and entail quick adaptations to a changing environment that only high-speed greedy heuristics can handle. These greedy heuristics consist in a permanent re-planning, considering the current system state. Intuitively, these quickly taken decisions are not appropriate for middle and/or long term planning due to the incessant erroneous movements.

However, we have demonstrated that the use of parallel memetic algorithms, which are more versatile than classical heuristics, can guide us towards more accurate solutions. With the intention of applying parallel memetic algorithms to such a dynamic environment, we have put forward a reformulation of the traditional problem of workforce distribution in dynamic multi-agent systems based on backboard architectures, which coalesces predictions of future system states with a precise search mechanism, by dynamically enlarging or diminishing the time-frame considered. We have claimed that the size of the time-frame depends upon the dynamism of the system (smaller when there is high dynamism and larger when there is low dynamism).

The present work has also illustrated how nearly optimal solutions each *v* seconds (size of the time-frame) outperforms continuous bad distributions when the right size of the time-frame is determined, and predictions and optimisations are correctly carried out. Particularly, we have proposed a neural network with an upgraded resilient propagation learning algorithm for predicting future system variables and a parallel memetic algorithm based on an island scheme to perform the assignment of incoming tasks to the right available agents.

Our approach has been tested out on a real-world production environment from Telefónica which is a large multinational telephone operator. We have shown that our approach not only outperforms other conventional techniques for separate but also as a unified technique. Therefore, we have obtained more accurate predictions than other famous forecast techniques for various problem instances. Besides, our search module based on a parallel memetic algorithm has outperformed other meta-heuristics under different scenarios. Additionally, the combination of the two modules with the adaptive middle-term time-frame has involved fine results. This corroborates that an adaptive middle-term time-frame can be a very powerful approach when having the required tools to implement it. But, all that glitters is not gold, and we assert our approach is not universal and might offer less accurate results than other approaches in environments in which timing is not a critical constraint or conditions are more stable and predictable.

Finally, the contributions to the scientific literature have produced the following peer-reviewed publications ((1) and (2) are less directly related to this dissertation):

1) Martínez-López, R.; Millán-Ruiz, D.; Martín-Domínguez, A. and Toro-Escudero, M.A.: *An Architecture for Next-Generation of Telecare Systems Using Ontologies, Rules Engines and Data Mining*. Proceedings of the International Conferences on Computational Intelligence for Modelling, Control and Automation; Intelligent Agents, Web Technologies and Internet Commerce; and Innovation in Software Engineering (CIMCA 2008), p. 31-36, Vienna, Austria, December 10-12, 2008.
2) Melendez, J.; López, B. and Millán-Ruiz, D.: *Probabilistic models to assist maintenance of multiple instruments*. Proceedings of the 14th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2009), p. 1499-1503, Palma de Mallorca, Spain, September 22-26th, 2009.
3) Pacheco, J.; Millán-Ruiz, D. y Vélez, J.L.: *Neural Networks for Forecasting in a Multi-skill Call Centre*. Proceedings of the 11th International Conference on Engineering Applications of Neural Networks (EANN 2009), p. 291-300, London, UK, August 27-29, 2009.

4) Millán-Ruiz, D. and Hidalgo, I.: *A Memetic Algorithm for Workforce Distribution in Dynamic Multi-Skil Call Centres*. Proceedings of the 10th European Conference on Evolutionary Computation in Combinatorial Optimisation (EVOCOP 2010), p. 178-189, Istanbul, Turkey, April 7-9, 2010.
5) Millán-Ruiz, D.; Pacheco, J.; Hidalgo, I. y Vélez, J.L.: *Forecasting in a Multi-skill Call Centre*. Proceedings of the 10th International Conference on Artificial Intelligence and Soft Computing (ICAISC 2010), Zakopane, Poland, June 13-17, 2010.
6) Millán-Ruiz, D. and Hidalgo, I.: *Algoritmo memético paralelo para la distribución de esfuerzo en centros de llamadas dinámicos multiagente y multitarea*. (Accepted) To appear in the 7th Spanish Conference on Meta-heuristics, Evolutionary Algorithms and Bioinspired Algorithms (MAEB 2010), Valencia, Spain, September, 2010.
7) Millán-Ruiz, D. and Hidalgo, I.: *Comparison of Metaheuristics for Workforce Distribution in Multi-Skill Call Centres*. Submitted to the International Joint Conference on Computational Intelligence (ICEC 2010).
8) Millán-Ruiz, D. and Hidalgo, I.: *A Self-Tuning Hybrid Memetic Algorithm for Dynamic Multi-Agent Systems based on Blackboard Architectures*. Submitted to the Workshop on Self-tuning, self-configuring and self-generating search heuristics (Self* 2010). Extended versions of selected contributions from this workshop will be considered for publication in a Special Issue of the Evolutionary Computation Journal, MIT Press.

7.2 AREAS OF FUTURE RESEARCH

To conclude, we propose some guidelines for future work. We recommend analysing more datasets and more problem instances because we may obtain different conclusions with regard to the configuration of our complete approach. So, if the arriving load is easy to predict we should choose a simpler forecast technique.

A deeper study on constraint handling should be done as our proposal is dependent on our specific domain (e.g. we may have different ranges for the levels of constraints).

For the dynamism levels, we can also have a continuous approximation (without levels) for those dynamic multi-agent systems where agents are not humans so that we do not need to care about the agents' rights (we can potentially change their profiles at any time without regulation constraints).

Additionally, we suggest that an analogous study for the search module comparison should be done, considering multi-objective evolutionary approximations (such as SPEA-II and NSGA-II), given our problem reformulation.

BIBLIOGRAPHY

[01] Baeza-Yates, R. and Ribeiro-Neto, B.: *Modern Information Retrieval*. Acm Press Series, Addison Wesley, 1999.

[02] Özsu, M. T. and Valduriez, P.: *Principles of Distributed Database Systems*. Second Edition, Prentice Hall, ISBN 0-13-659707-6, 1999.

[03] Andrews, G.R.: *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison–Wesley, ISBN 0-201-35752-6, 2000.

[04] Chen, Z.; Yang, M.; Francia, G. and Dongarra, J.: *Self Adaptive Application Level Fault Tolerance for Parallel and Distributed Computing*. ipdps, pp.414, IEEE International Parallel and Distributed Processing Symposium, 2007.

[05] Asiki, A.; Tsoumakos, D. and Koziris, N.: *An Adaptive On-line System for Efficient Processing of Hierarchical Data*. Proceedings of the 18th International ACM Symposium on High Performance Distributed Computing (HPDC'09), Garching, Germany, 2009.

[06] Erman, L.; Hayes-Roth, F.; Lesser, V.R. and Reddy, D.R.: *The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty*. Computing Surveys, 12(2):213-253, 1980.

[07] Hayes-Roth, B.: *A blackboard architecture for control*. Artificial Intelligence, pp. 251-321, 1985.

[08] Avramidis, A.N.; Chan, W.; Gendreau, M.; L'Ecuyer, P. and Pisacane, O.: *Optimizing daily agent scheduling in a multiskill CC*. European Journal of Operational Research (2009).

[09] Brucker, P.: *Scheduling algorithms*. 2nd edn. Springer, Heidelberg, 1998.

[10] Chauvet, F.; Proth, J.M. and Soumare, A.: *The simple and multiple job assignment problems*. International Journal of Production Research 38(14), 3165–3179, 2000.

[11] Bhulaii, S.; Koole, G. and Pot, A.: *Simple Methods for Shift Scheduling in Multiskill Call Centers*. M&SOM 10(3), 411–420, 2008.

[12] Koole, G.: *Call Center Mathematics: A scientific method for understanding and improving contact centers*, http://www.cs.vu.nl/~koole/ccmath/book.pdf , 2006.

[13] Garey, M. and Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman. ISBN 0-7167-1045-5, 1979.

[14] Alberts, L.J.: *Churn prediction in the mobile telecommunication industry: An application of Survival Analysis in Data Mining*. http://www.personeel.unimaas.nl/westra/PhDMaBa-teaching/GraduationStudents/LaurensAlberts2006/Presentatie.ppt ,2006.

[15] Dasgupta, K.; Singh, R.; Viswanathan, B.; Chakraborty, D.; Mukherjea, S.; Nanavati, A. and Joshi, A.: *Social ties and their relevance to churn in mobile telecoms networks*. EDBT'08. March 25-30, Nantes, France, 2008.

[16] Gouarderes, G.; Minko, A. and Richard, L.: *Simulation and multi-agent environment for aircraft maintenance learning*. AIMSA 2000, Varna , vol. 1904, pp. 152-166, ISBN 3-540-41044, 2000.

[17] Guttman, R.H. and Maes, P.: *Agent-mediated Integrative Negotiation for Retail Electronic Commerce*. Proceedings of the Workshop on Agent Mediated Electronic Trading (AMET'98), Minneapolis, Minnesota, pp. 70-80, 1998.

[18] Massaguer, D.; Balasubramanian, V.; Mehrotra, S. and Venkatasubramanian, N.: *MultiAgent Simulation of Disaster Response*. ATDM Workshop in AAMAS, 2006.

[19] Rouhana, N. and Horlait, E.: *Dynamic Congestion Avoidance Using Multi-Agents Systems*. Lecture Notes In Computer Science, Vol. 2164, Proceedings of the Third International Workshop on Mobile Agents for Telecommunication Applications, 2001.

[20] Timofeev, A.V.; Syrtzev, A.V. and Kolotaev, A.V.: *Network analysis, adaptive control and imitation simulation for multi-agent telecommunication systems*. Physics and Control, Proceedings International Conference, ISBN 0-7803-9235-3, 2005.

[21] Franklin, S. and Graesser, A.: *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

[22] Russell, S.J. and Norvig, P.: *Artificial Intelligence: A Modern Approach*. 2nd ed. Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2, chapter 2, 2003.

[23] Kasabov, N.: *Introduction: Hybrid intelligent adaptive systems*. International Journal of Intelligent Systems, Vol.6, 453-454, 1998.

[24] Wooldridge, M.: *An Introduction to MultiAgent Systems*, John Wiley & Sons Ltd, paperback, 366 pages, ISBN 0-471-49691-X, 2002.

[25] Dechter, R.: *Constraint Processing*. Morgan Kaufmann. ISBN 1-55860-890-7, 2003.

[26] Hoos, H.H. and Stutzle, T.: *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2005.

[27] Wil Michiels, E.A. and Korst, J.: *Theoretical Aspects of Local Search (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer 1 edition, ISBN-10: 3540358536, 2007.

[28] Holland, J.H.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[29] Goldberg, D. E.: *Genetic Algorithms in Search Optimization and Machine Learning*. Addison Wesley. pp. 41, 1989.

[30] Moscato, P.: *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*. Caltech Concurrent Computation Program (report 826), 1989.

[31] Lai, T. L. and Robbins, H.: *Optimism in the face of uncertainty: Asymptotically efficient adaptive allocation rules*. Advances in Applied Mathematics, 6:4–22, 1985.

[32] Auer, P.; Cesa-Bianchi, N. and Fischer, P.: *Finite time analysis of the multiarmed bandit problem*. Machine Learning, 47(2-3) pp. 235–256, 2002.

[33] Audibert, J.; Munos, R. and Szepesvári, C.: *Tuning bandit algorithms in stochastic environments*. ALT, 2007.

[34] Harchol-Balter, M. and Downey, A.: *Exploiting process lifetime distributions for dynamic load balancing*. ACM Transactions on Computer Systems, vol. 15, pp. 253–285, 1997.

[35] Hansen, P. and Mladenovic, N.: *Variable neighbourhood search: Principles and applications*. European Journal of Operations Research, 130:449–467, 2001.

[36] Caragiannis, I.; Flammini, M.; Kaklamanis, C.; Kanellopoulos, P. and Moscardelli, L.*: Tight bounds for selfish and greedy load balancing*. ICALP, LNCS, 2006.

[37] Gans, N., Koole, G. and Mandelbaum, A.: *Telephone Call Centers: a Tutorial and Literature.* Review. 2, 2002.

[38] Messerli, E.J.: *Proof of a convexity property of the Erlang B formula*. Bell System Technical Journal 51, 951-953, 1972.

[39] Inayatullah, M.; Ullah, F.K.; Khan., A.N.: *An Automated Grade Of Service Measuring System.* IEEE—ICET 2006, 2nd International Conference on Emerging Technologies, Peshawar, Pakistan 13-14, pp.230-237, 2006.

[40] Mandelbaum, A. and Zeltyn, S.: *The Palm/Erlang-A Queue, with Applications to Call Centers.* Advances in Services Innovations, 2005.

[41] Eddy, S. R.: *What is dynamic programming?* Nature Biotechnology, 22, 909-910, 2004.

[42] Lawler, E. L. and Wood, D.E.: *Branch-and-bound methods: A survey*. Operations Research, 14(4):699--719, 1966.

[43] Aouchiche, M.; Caporossi, G. and Hansen, P.: *Variable neighborhood search for extremal graphs: Variations on graffiti*. Congressus Numerantium, 148:129–144, 2001.

[44] Den Besten, M. and Stützle, T.: *Neighborhoods revisited: Investigation into the effectiveness of variable neighborhood descent for scheduling*. In MIC'2001, pages 545–549, Porto, 2001.

[45] Kirkpatrick, S.; Gelatt, C. D. and Vecchi, M.P.: *Optimization by Simulated Annealing*. Science. New Series 220 (4598): 671–680, 1983.

[46] Glover, F. and M. Laguna: *Tabu Search*. Kluwer, Norwell, MA, 1997.

[47] Glover, F.: *Tabu Search — Part I*, ORSA Journal on Computing, 190-206, 1989.

[48] Glover, F. *Tabu Search — Part II*, ORSA Journal on Computing, 4-32, 1990.

[49] Cvijovic, D.; Klinowski, J.: *Taboo search - an approach to the multiple minima problem*. Science, 664-666, 1995.

[50] Glover F.: *A Template for Scatter Search and Path Relinking*. Lecture Notes in Computer Science, 1363: 1-53, 1997.

[51] Katayama, K. and Narihisa, H.: *Iterated local search approach using genetic transformation to the traveling salesman problem*. Proceedings of GECCO'99, Vol. 1. Morgan Kaufmann, pp. 321–328, 1999.

[52] Kocsis L. and György A.: *Efficient Multi-start Strategies for Local Search Algorithms*. LNCS, Machine Learning and Knowledge Discovery in Databases, 705-720, 2009.

[53] Festa, P. and Resende, M.G.C.: *GRASP: An annotated bibliography*. Essays and Surveys on Metaheuristics, pp. 325–367, Kluwer Academic Publishers, 2002.

[54] Dorigo, M.: *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italie, 1992.

[55] Kennedy, J. and Eberhart, R.: *Particle Swarm Optimization*. Proceedings of IEEE International Conference on Neural Networks, pp. 1942-1948, 1994.

[56] Di Caro, G. and Dorigo, M.: *Mobile agents for adaptive routing*. In H. El-Rewini, editor, Proceedings of the 31st International Conference on System Sciences (HICSS-31), pages 74–83. IEEE Computer Society Press, Los Alamitos, CA, 1998.

[57] Meyn, S.P. and Tweedie, R.L.: *Markov Chains and Stochastic Stability*. London: Springer-Verlag, 1993.

[58] Fang, Y.: *Hyper-Erlang Distribution Model and its Application in Wireless Mobile Networks*. Wireless Networks, Springer Netherlands, ISSN 1022-0038, Volume 7, Number 3, 2001.

[59] Dutta, M. and Chaubey, V.K.: *Performance Analysis of All-Optical WDM Network with Wavelength Converter Using Erlang C Traffic Model*. Communications in Computer and Information Science, Information Processing and Management, Springer, ISBN 978-3-642-12213-2, pp. 238-244, 2010.

[60] Gallager, R.G.: *Discrete stochastic processes*. Kluwer, Boston, 1996.

[61] Lindley, D.V.: *Regression and correlation analysis*. New Palgrave: A Dictionary of Economics, v. 4, pp. 120–23, 1987.

[62] Wei, W.W.: *Time series analysis: Univariate and multivariate methods*. New York: Addison-Wesley, 1989.

[63] Zheng, Y. and Xu, R.: *An Adaptive Exponential Smoothing Approach for Software Reliability Prediction*. Wireless Communications, Networking and Mobile Computing, 2008.

[64] Bowerman, B.L. and O'Connell, R.T.: *Forecasting and Time Series: An Applied Approach*. Duxbury classic series, third edition, 1993.

[65] Makridakis, S.; Wheelwright, S.C. and HYNDMAN, R.J.: *Forecasting: Methods and Applications*. John Wiley and Sons, Inc., 1998.

[66] Antipov, A. and Meade, N.: *Forecasting call frequency at a financial services call centre*. Journal of operation research, pp. 953-960, 2002.

[67] Van den Bergh, K.: *Predicting Call Arrivals in Call Centers*. Website: http://www.few.vu.nl/stagebureau/werkstuk/werkstukken/werkstuk-bergh.pdf, thesis, 2006.

[68] Cotez, P.; Rio, M.; Rocha, M. and Sousa, P.: *Internet Traffic Forecasting using Neural Networks*. International Joint Conference on Neural Networks, 2006.

[69] Zaiyong, T. and Fishwick, P.A.: *Feed-forward Neural Nets as Models for Time Series Forecasting*. ORSA Journal of Computing, 1993.

[70] Mandic, D. and Chambers, J.: *Recurrent Neural Networks for Prediction: Architectures, Learning algorithms and Stability*, Wiley, 2001.

[71] A. B. Shabri: *Comparision of Time Series Forecasting Methods Using Neural Networks and Box-Jenkins Model*. Matematika, ISSN 01278274, 2001.

[72] Michie, D.; Spiegelhalter, D.J. and Taylor, C.C.: Machine Learning, Neural and Statistical Classification, 1994.

[73] Phansalkar, V. and Sastry, P.S.: *Analysis of the Back-Propagation Algorithm with Momentum*. IEEE Transactions on Neural Networks, pp. 505-506, 1994.

[74] Igel, C. and Hüsken, M.: *Empirical Evaluation of the Improved Rprop Learning*. Neurocomputing 50, 105-123, 2003.

[75] Aquino, I.; Perez, C.; Chavez, J.K. and Oporto, S.: *Daily Load Forecasting Using Quick Propagation Neural Network with a Special Holiday Encoding*. Neural Networks, 2007. IJCNN 2007. International Joint Conference on Neural Networks, 2007.

[76] Luenberger, D. G. and Ye, Y.: *Linear and nonlinear programming*. International Series in Operations Research & Management Science, New York, Springer, pp. xiv+546, 2008.

[77] Buhmann, M. D.: *Radial Basis Functions: Theory and Implementations*. Cambridge University Press, ISBN 978-0-521-63338-3, 2003.

[78] Fahlman, S. and Lebiere, C.: *The Cascade-Correlation Learning Architecture*. Article created for National Science Foundation under Contract Number EET-8716324 and Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, 1991.

[79] Hopfield, J.J.: *Neural networks and physical systems with emergent collective computational abilities.* Proceedings of the National Academy of Sciences of the USA, vol. 79, no. 8, pp. 2554-2558, 1982.

[80] Haykin, S.: *Neural Networks: A Comprehensive Foundation*. Second edition, Prentice Hall, ISBN 0132733501, 1998.

[81] Kingsland, S. E.: *Modeling nature: episodes in the history of population ecology*. Chicago: University of Chicago Press, ISBN 0-226-43728-0, 1995.

[82] Cadieu C.; Kouh M.; Pasupathy A.; Conner C.E.; Riesenhuber M. and Poggio T.: *A Model of V4 Shape Selectivity and Invariance*. J Neurophysiol 98: 1733-1750, 2007.

[83] Lindeberg, T.: *Scale-space for discrete signals*. PAMI(12), No. 3, pp. 234-254, 1990.

[84] Osborn, G.: *Mnemonic for hyperbolic formulae*. The Mathematical Gazette, p. 189, volume 2, issue 34, 1902.

[85] Sagias, N. C. and Karagiannidis, G. K.: *Gaussian class multivariate Weibull distributions: theory and applications in fading channels*. Institute of Electrical and Electronics Engineers. Transactions on Information Theory 51 (10): 3608–3619, 2005.

[86] Riedmiller, M. and Braun, H.: *Rprop - A Fast Adaptive Learning Algorithm*. Proceedings of the International Symposium on Computer and Information Science VII, 1992.

[87] Bishop, C.M.: *Neural Networks for Pattern Recognition.* Oxford University Press. ISBN 0-19-853849-9, 1995.

[88] Silva, F.M. and Almeida, L.B.: *Speeding up backpropagation*. Advanced Neural Computers, pages 151–158. North-Holland, 1990.

[89] Igel, C. and Hüsken, M.: *Improving the Rprop Learning Algorithm*. Proceedings of the Second International Symposium on Neural Computation, NC'2000, pp. 115–121, ICSC Academic Press, 2000.

[90] Jacobs, R.A.: *Increased rates of convergence through learning rate adaptation*. Neural Networks, 1(4):295–307, 1988.

[91] LeCun, Y.; Bottou, L.; Orr, G.B. and Müller, K.R.: *Efficient backprop*. Neural Networks: Tricks of the Trade, number 1524 in LNCS, chapter 1. Springer-Verlag, 1998.

[92] Tollenaere, T.: *Supersab: Fast adaptive backpropagation with good scaling properties*. Neural Net-works, 3:561–573, 1990.

[93] Miller, B.L. and Goldberg, D.: *Genetic Algorithms, Tournament Selection, and the Effects of Noise.* Report 950006, 1995.

[94] Muhlenbein, H. and Schlierkamp-Voosen, D.: *Predictive Models for the Breeder Genetic Algorithm*. Evolutionary Computation, 1993.

[95] Wills, C.: *Rank-Order Selection Is Capable of Maintaining All Genetic Polymorphisms*. Genetics. 89(2): 403–417, 1978.

[96] Gayon, J. and Matthew, C.: *Darwinism's Struggle for Survival: Heredity and the Hypothesis of Natural Selection*. Cambridge University Press, pp. 158, 1998.

[97] Gog, A.; Chira, C.; Dumitrescu, D. and Zaharie, D.: *Analysis of Some Mating and Collaboration Strategies in Evolutionary Algorithms*. 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC '08, 2008.

[98] Ishibuchi, H. and Shibata, Y.: *A Similarity-based Mating Scheme for Evolutionary Multiobjective Optimization*. Lecture Notes in Computer Science, 2003.

[99] Gwiazda, T.: *Genetic Algorithms Reference Vol.1 Crossover for single-objective numerical optimization problems*. Lomianki, 2006.

[100] Spears, W. and De Jong, K.: *An Analysis of Multi-Point Crossover*. Proc. Foundations of Genetic Algorithms Workshop, 1990.

[101] Sywerda, G.: *Uniform crossover in genetic algorithms*. Proceedings of the third international conference on Genetic algorithms, George Mason University, United States, pp. 2-9, 1989.

[102] Vavak , F. and Fogarty, T.C.: *Comparison of Steady State and Generational Genetic Algorithms for Use in Nonstationary Environments*. IEEE International Conference on Evolutionary Computation (ICEC), 1996.

[103] Chakraborty, B. and Chaudhuri, P.: *On The Use of Genetic Algorithm with Elitism in Robust and Nonparametric Multivariate Analysis*. Austrian Journal of Statistics, Volume 32, Number 1&2, pp. 13–27, 2003.

[104] Koole, G.; Pot, S. and Talim, J.: *Routing heuristics for multi-skill call centers*. Proceedings of the Winter Simulation Conference, 1813–1816, 2003.

[105] Whitt W.: *Staffing a call center with uncertain arrival rate and absenteeism*. PO&M, 2006.

[106] Garnett, O. and Mandelbaum, A.: *An Introduction to Skills-Based Routing and its Operational Complexities*. Teaching Note, 2000.

[107] Thompson, G. M.: *Labor staffing and scheduling models for controlling service levels*. Naval Res. Logist, 719–740, 1997.

[108] Ingolfsson, A.; Cabral, E. and Wu, X.: *Combining integer programming and the randomization method to schedule employees*. TR, School of Business, University of Alberta, 2007.

[109] Land, A.H. and Doig, A.G.: *An automated method for solving discrete programming problems*. Econometrica 28 497–520, 1960.

[110] Gomory, R.E.: *Outline of an algorithm for integer solutions to linear programs*. Bull. Amer. Math. Soc. 64 275–278, 1958.

[111] Ahrens, J.H. and Ulrich, D.: *Computer Methods for Sampling from Gamma, Beta, Poisson and Binomial Distributions*. Computing 12 (3): 223-246, 1974.

[112] Micheli, A.: *Neural Network for Graphs: A Contextual Constructive Approach*. IEEE Transactions on Neural Networks, 20:3, 498-511, 2009.

[113] Jolliffe, I.T.: *Principal Component Analysis*. Series: Springer Series in Statistics, 2nd ed., Springer, NY, XXIX, 487 p. 28 illus, ISBN 978-0-387-95442-4, 2002.

[114] http://www.cs.waikato.ac.nz/ml/weka/

[115] Kohavi, R.: *A study of cross-validation and bootstrap for accuracy estimation and model selection*. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence 2 (12): 1137–1143, 1995.

[116] http://robjhyndman.com/software/forecast