Simon Martin

Multi-Agent based cooperative search in combinatorial optimisation

The thesis is submitted in partial fulfilment

of the requirements for the award of the

degree of

Doctor of Philosophy
of the University of Portsmouth.

First Supervisor:

Dr. Djamila Ouelhadj

Second Supervisor:

Dr. Dylan Jones

February 5, 2013

DECLARATION

Whilst registered as a candidate for the above degree, I have not been registered for any other research award. The results and conclusions embodied in this thesis are the work of the named candidate and have not been submitted for any other academic award.

DEDICATION

To my wife and children without whose patience and understanding I would not have completed this thesis.

ACKNOWLEDGEMENTS

I would like to thank Dr Djamila Ouelhadj my first supervisor for her help and encouragement throughout this PhD. Also I would like to acknowledge the contributions of Dr Dylan Jones and Dr Ender Özcan. For help in proof reading this thesis, I would like to thank my mother Valerie Martin and my wife Claudia Wittum.

ABSTRACT

Cooperative search provides a class of strategies to design more effective search methodologies by combining (meta-) heuristics for solving combinatorial optimisation problems. This area has been little explored in operational research. This thesis proposes a general agent-based distributed framework where each agent implements a (meta-) heuristic. An agent continuously adapts itself during the search process using a cooperation protocol based on reinforcement learning and pattern matching. Good patterns which make up improving solutions are identified and shared by the agents. A theoretical approach to the understanding of the potential of agent-based systems is also proposed. This agent-based system aims to raise the level of generality by providing a flexible framework to deal with a variety of different problem domains. The proposed framework so far has been tested on Permutation Flow-shop Scheduling, Travelling Salesman Problem and Nurse Rostering. These instances have yielded some promising results. As part of the nurse rostering work a novel approach to modelling fairer nurse rosters is proposed.

Contents

List of Figures

List of Tables

DISSEMINATION

Technical Reports

(Soon the be submitted to journals)

Martin S., Ouelhadj, D., Beullens P., Ozcan E. (2012) A generic agent-based framework for cooperative search using pattern matching and reinforcement learning Technical Report. University of Portsmouth.

Ouelhadj Djamila, Martin Simon, Smet, P., Ozcan E. and Vanden Berghe G. (2012) Fairness in nurse rostering. Technical Report. University of Portsmouth.

Conference proceedings

Smet P., Martin Simon, Ouelhadj Djamila, Ozcan E. and Vanden Berghe G. (2012) Investigation of fairness measures for nurse rostering. In: 9th International Conference on the Practice and Theory of Timetabling (PATAT 2012), 28-31 August 2012, Norway.(p369-373)

Martin S., Ouelhadj, D., Beullens P., Ozcan E. (2010) An agent-based framework for cooperative hybrid/ meta-heuristic search for TSP. Presented at Workshop on Systems to Build Systems, IEEE World Congress on Computational Intelligence(2010). Barcelona.

Conferences

Martin Simon,Smet P., Ouelhadj Djamila, Ozcan E. and Vanden Berghe G. (2012)

Agent-based Cooperative Meta-heuristic search for Fairness in Nurse Rostering. EURO 2012 July Vilnius Lithuania.

Martin S., Ouelhadj D., Beullens P., Ozcan E. (2010) Multi-Agent search. OR53 Annual Conference 2011 Nottingham.

Martin S., Ouelhadj D., Beullens P., Ozcan E. (2010) A generic agent-based framework for cooperative hybrid meta-heuristic search. EURO 2010 Lisbon.

CHAPTER I
INTRODUCTION

1.1 Background and Motivation

Heuristics (meta-heuristics) have been successfully used to solve a wide range of combinatorial optimisation problems. In the recent years, however, it has become evident that different (meta-)heuristics working on the same problem can produce different results. Moreover, most of the (meta-)heuristics developed for a specific problem domain cannot be used to solve instances from another problem domain. This frequently requires either parameter tuning and/or design of new neighbourhood operators for the new problem domain. There is almost no guidance available in choosing the best (meta-)heuristic for solving a problem in hand. For these reasons, the use of a sole (meta-)heuristic can be rather restrictive when dealing with real-world problems. But what if there was a way of combining all of these meta-heuristics so that the different strengths could be harnessed during a search?

One way that this could be achieved is with a framework enabling the use of different (meta-)heuristics that to provide an improved search methodology and an increase in the level of generality. The key idea behind cooperative search is to combine the strengths of different (meta-)heuristics to balance intensification and diversification and direct the search towards promising regions of the search space (Ouelhadj and Petrovic, 2010).

Interest in cooperative search has risen due to successes in combining novel search algorithms (Clearwater et al., 1992; Hogg and Williams, 1993; Talbi and Bachelet, 2006). Blum and Roli (2003); Clearwater et al. (1992); Hogg and Williams (1993); Toulouse et al. (1999); Crainic and Toulouse (2008) describe how cooperative search can be performed by the exchange of states, solutions, sub-problems, models or search space characteristics. Several frameworks have been proposed recently, including (Talbi and Bachelet, 2006; Milano and Roli, 2004; Meignan et al., 2008, 2010; Ouelhadj and Petrovic, 2010). Each of these frameworks incorporates either meta-heuristics

(Talbi and Bachelet, 2006; Milano and Roli, 2004) or hyper-heuristics (Ouelhadj and Petrovic, 2010).

Crainic and Toulouse (2008) explain that cooperation may take many forms but in each case they share two important features:

a) a set of *autonomous programmes (AP)* implementing a particular solution method,

b) a cooperation scheme for combining AP's into a single problem-solving strategy.

Another important feature that will be developed in chapter IV is that AP's can just as well solve a problem as stand alone algorithms. However by cooperating the chances of finding novel and greatly improved solutions are increased. Therefore the communication and sharing of information is an important feature of cooperation. Crainic and Toulouse (2008) have identified a number of key properties that might feature in a cooperation schema. These are synchronous, asynchronous, direct and indirect communication. Cooperation mechanisms where AP's have to synchronise communication at predefined intervals are prone to be slow and less affective as the search has to be re-started at each interval. Asynchronous mechanisms do not have such constraints therefore information exchange can take place in a more seamless manner allowing a search to expand in hitherto unexpected ways. Direct communication is often associated with population based methods where are population is divided into subsets or an *island* which an communicate directly with other islands. Indirect communication is often associated with memory-based approaches where AP's send and receive partial or whole solutions to a central memory pool of information.

1.1.1 Motivation

In the literature most cooperative search mechanisms are described as communicating indirectly through some central pool or adaptive memory (Talbi and Bachelet, 2006; Milano and Roli, 2004; Meignan et al., 2008). This can take the form of passing

whole, or possibly, partial solutions, to the pool (Meignan et al., 2010). As far as it is known direct asynchronous cooperation has not been researched much at all.

The only exceptions are Vallada and Ruiz (2009) and Ouelhadj and Petrovic (2010) where whole solutions are passed from one process to another in an island model executing a genetic algorithm (Vallada and Ruiz, 2009) or hyper-heuristic (Ouelhadj and Petrovic) to solve the permutation flow-shop scheduling problem. Also Xie and Liu (2009) propose an evolutionary system to solve the Travelling Salesman Problem (TSP). However, all three systems are designed for specific problem domains and use the best known (meta-)heuristics for these domains. While Malek (2010) has proposed a multi-agent framework able to work on different problem domains, but the system seems to pass whole solutions through a solution pool.

Little work has been done on direct cooperation where partial solutions are rated and their parameters are communicated between agents. Furthermore no direct cooperation strategy has been applied to more than one problem domain in combinatorial optimisation. There is a gap in the literature regarding agents cooperating directly and asynchronously where the communication is used for the adaptive selection of moves with parameters. Also little work seems to have been done on cooperative systems where different meta-heuristics work collectively to solve problems.

1.2 Aims, objectives and contribution

The aim of this thesis is to find ways to combine different meta-heuristics in such a way that they cooperate with each other. This goal itself necessitates a number of design choices. According to Crainic and Toulouse (2008) an asynchronous framework enabling the use of different (meta-)heuristics could result in an improved search methodology and increase the level of generality. Furthermore using different cooperating meta-heuristics also implies a choice between direct or indirect methods of communication where each meta-heuristic is an island. Given these choices Crainic and Toulouse (2008) suggest that communication can either be many-to-many where each meta-heuristic communicates with every other, or it can be memory based where

information is sent to a pool where the other meta-heuristics can use it as required. However, one of the aims of this project is to build a framework using different meta-heuristics to solve combinatorial optimisation problems balancing intensification and diversification between (meta-heuristics). It is proposed that this goal is appropriately achieved by developing a multi-agent framework where the agents are autonomous and maintain there own representation of the a search environment. This, in turn, necessitates an asynchronous direct many-to-many approach which, as far as is known, has not been tried before.

Given this set of assumptions the aims for the thesis are as follows:

a) develop a theory to prove why cooperative algorithms are effective

b) build a system where meta-heuristics share information using asynchronous direct cooperation

c) develop a multi-agent platform for cooperative search that is generic and modular that can be easily configured to solve different combinatorial optimisation problems

d) develop a cooperation mechanism that allows different meta-heuristics to share good parts of solutions with each other to control intensification and diversification effectively.

e) test the framework with a number of case studies to prove that the system meets its design objectives.

To achieve these goals a generic framework of cooperating agents using pattern matching and reinforcement learning is proposed. Therefore an island model is proposed where each agent is autonomous and is capable of executing different meta-heuristic and local search combinations with different parameter settings. They cooperate using an asynchronous message passing protocol utilising a pattern matching diversification phase and meta-heuristics for intensification.

To this end, the framework has been developed to use ontologies (see section 5.4.1.1) to model combinatorial optimisation problems. In so doing the platform is easily configurable, generic and modular allowing the seamless integration of new heuristics. It also means that different modules developed by other people can be easily adapted and added to the platform. Finally, the system has been tested on a number of classical combinatorial optimisation problems including the Symmetrical Travelling Salesman Problem(STSP) and the Permutation Flow-shop Scheduling Problem (PFSP). The system was also tested on a highly constrained Nurse Rostering (NR) problem.

1.2.1 Contribution

The main contributions of this thesis are:

- A Turing machine based theory of cooperating algorithms. Cooperation is examined within the context of the theory of computation and computability. To account for cooperation formally an extension to the definition Turing machine is proposed. The formal implications of this work are discussed.

- A generic modular agent-based framework for cooperative search to combinatorial optimisation problems is proposed and implemented. The framework raises the level of generality.

- A generic pattern matching protocol for meta-heuristics to share information is proposed. Each meta-heuristic is an autonomous agent with its own representation of the search environment. To this end they share partial solution pattens to enable each agent to build new potential solutions and develop the search.

- Important test results showing the benefits of cooperation for The Symmetrical Travelling Salesman Problem (STSP), the Permutation Flowshop Scheduling Problem (PFSP) and the Nurse Rostering Problem (NRP).

- A Nurse rostering model is used by the framework to produce test results that out perform previous work on these problem instances. This work has been conducted in conjunction with (Ouelhadj et al., 2012)

- A novel approach to fairness in nurse rostering. With Ouelhadj et al. (2012) a new objective function is proposed to increase the level of fairness in nurse rosters. This was tested on the framework with good results.

1.3 Outline of the thesis

This thesis is structured as follows. A brief overview of the literature is provided in chapter II. It reviews the current research on parallel meta-heuristics and agent-based cooperation. It pinpoints the gap in the literature which the current work seeks to fill.

Agents and agent-based systems are introduced in chapter III. It starts with a brief history of agent-based research and concludes with a discussion of the JADE platform which was used to develop the agents-based system described in this thesis.

In chapter IV A Turing machine definition of cooperating algorithms is proposed. It identifies them as computing a class of algorithms proposed by Crainic and Toulouse (2008) where, if required, algorithms can solve given problems on their own, but crucially, if they are allowed to cooperate, new and unforeseen solutions are generated. This class of algorithms lends itself well to the goals of this research, namely to get meta-heuristics to cooperate asynchronously to improve solution quality. The implications of this class of algorithms for the theory of computation and computability are examined and it is proposed that it is always possible to extend the Turing machines by augmenting a machines alphabet or the set of tape head moves, but that this does not affect the Church-Turing thesis as has been claimed by Goldin (2000).

Chapter V the multi-agent framework of combinatorial optimisation problems is introduced. The cooperation protocol for pattern matching is described as well as the ontology representing combinatorial optimisation. This allows the system to operate

at a level of generality making it possible to work on different combinatorial optimisation problems with little configuration. Section 5.2 discusses the implementation of the generic agent-based system.

Chapter VI is the first of the case studies where the platform is tested on well known benchmark problems for the Symmetrical Travelling Salesman Problem (STSP). The framework is tested using two simple meta-heuristics combined with a 2-opt local search heuristic. Even with such simple heuristics the system produces credible results.

Chapter VII is the second of the case studies for the Permutation Flow-shop Scheduling Problem (PFSP). The system uses the same meta-heuristics and parameter setting as with the STSP tests. The only difference is that the local search heuristic is a simple hill-climber with random swaps. The test results are in line with many published results and are surprisingly good given the simplicity of the heuristics.

Chapter VIII is the last of the case studies testing the platform on a constrained problem, The Nurse Rostering problem (NR). The system is tested on benchmark problems from a hospital in Belgium (Bilgin, 2008). These results are better than those published in Bilgin et al. (2012). Work was also carried out with Belgian partners (Ouelhadj et al., 2012) on fairness in nurse rostering where a new objective function modelling fairness was tested.

The final chapter IX summarises the main achievements of the thesis, presents general conclusions and suggests future research directions for cooperation in combinatorial optimisation.

CHAPTER II
A SURVEY OF COOPERATIVE SEARCH

2.1 Introduction

Research into parallel cooperative optimisation has grown significantly in the last 10 to 15 years. Most of this work has been focussed on parallel (meta-)heuristics, including tabu search, genetic algorithms,ant colony and simulated annealing (Crainic and Laporte, 1998; Crainic and Toulouse, 2010; Alba, 2005; Dorigo and Gambardella, 1997; Dorigo et al., 2006; Aydin, 2007). This research has concentrated on speed-up and robustness. Speed-up aims to decrease the overall processing time of (meta-)heuristics by implementing them in parallel rather than a single process implementation. This can be achieved by processing computationally expensive or time consuming routines in parallel. Robustness is where the overall search is widened covering more of the search space by starting different processes with different instances of the same problem without the need for parameter tuning.

There is very little literature on cooperating meta-heuristic agents. This chapter reviews the work on parallel meta-heuristics and shows why cooperating parallel meta-heuristics are different from cooperating meta-heuristic agents.

2.2 Brief overview of Parallel Meta-heuristics

Meta-heuristics are notoriously difficult to calibrate, and in consequence they have to be designed and tuned to specific problems and cannot be easily reconfigured to solve new problems. However parallel meta-heuristics have been shown to address some of these issues. Crainic and Toulouse (2010) identify four main areas of research in the parallel meta-heuristics field:

- **Low-level parallel strategies.** These usually involve the decomposition of a task into subtasks where a master process apportions subtasks to slave processes. The slaves complete their tasks and return the results to the master

process which recombines the results either producing a final result or reiterating the whole process again. There is no interprocess communication between slaves. This is a very traditional approach to a parallelism and is typical of research undertaken in the 1980's such as Malek et al. (1989).

- **Domain Decomposition.** The idea here is to break a problem into smaller, usually disjoint, but not necessarily exhaustive subsets, and try to solve these with (meta-)heuristics and then to collect the respective partial solutions and reconstruct the entire one. Laganiére and Mitiche (1995) propose a parallel tabu search using domain decomposition to solve the vehicle routing problem with time windows.

- **Independent multi-search.** This strategy involves the performing of several searches simultaneously. There is no master process that decomposes a problem or one that controls the search. Starting from different initial solutions the entire search space is searched in parallel by different search algorithms. The best result is collected and presented as the overall solution at the end of the search. An example of this has been proposed by Taillard (1994).

- **Cooperative multi-search.** Cooperative multi-search starts in much the same way as independent multi-search strategies with parallel processes starting from the same or different seed solutions. However the difference is that during the search the various processes have the opportunity to share information. This can be done synchronously or asynchronously. In the former case, the processes have to be halted periodically throughout the search while the different processes share information with each other updating the overall system state. In the case of asynchronous cooperation, the different processes control how they share information with each other. This latter method tends include seamless and more effective information sharing. For example, consider the work of Vallada and Ruiz (2009) where different islands in a genetic algorithm update asynchronous each other asynchronously through a central memory pool.

Crainic et al. (2005) propose a classification system where they identify a three of important dimensions of parallel meta-heuristics. These include:

- **Search Control Cardinality.** This specifies if the global control strategy for the parallel system is controlled by one (1C) or many (pC)processes.

- **Search Control and Communications.** This addresses the issue of whether information is to be passed synchronously or asynchronously. In the former case, some process has to control when information is to be passed between processes. This usually results in all processes being halted while this exchange takes place. In the later case information exchange is controlled by each process. There are also four sub-categories associated with this category. These are: rigid synchronous(RS), knowledge synchronisation(KS), and collegial(C) and knowledge collegial(KC). Here asynchronous information exchange is *collegial* because no process has overall control.

- **Search Differentiation.** Different processes/threads can start from the same or different seeds. Furthermore, the different threads/processes can use the same or different search strategies. These choices are split into four sub categories. These are: same initial point/population, same search strategy(SPSS); same initial point/population, different search strategy(SPDS); Multiple initial points/populations, same search strategy(MPSS); Multiple initial points/populations, different search strategy(MPDS). The term "point" is concerned with neighbourhood-based methods while "population" is associated with evolutionary of genetic based methods

Taking into account the above categories, it is possible to classify most parallel heuristics seen in the literature. Furthermore it is possible to classify cooperation strategies and pinpoint the approach chosen in this thesis.

2.2.1 Cooperative meta-heuristics

Using the classification schema of Crainic et al. (2005), it is clear that all cooperative meta-heuristics fall into the category of pC/KS or pC/C with any of the search differentiation subcategories(SPSS/SPDS/MPSS/MPDS).

The main challenge for cooperative search is designing cooperative mechanisms that enable the useful exchange of information between processes. This cooperation may be performed synchronously or asynchronously, or directly or indirectly (Crainic and Toulouse, 2008). An example of direct cooperation is the island model of evolutionary algorithms, where a population is divided into subsets, each is assigned to a different processor and a genetic algorithm runs on each island. The islands may communicate with each other. This normally takes the form of the islands sending good solutions to a pool of solutions. Solutions are then retrieved from the pool for further search. This model has been successfully applied to a number of combinatorial optimisation problems. Crainic et al. (1995a) apply a pC/KS/MPSS approach to multi-commodity location with balancing requirements, while Crainic et al. (2006) use a pC/C/MPSS approach on the capacitated network design problem,James et al. (2009) uses a pC/KC/MPSS strategy for quadratic assignments and Crainic et al. (2009) use a pC/KC/MPDS approach for a methodology for designing wireless networks.

Comparative literature studies conducted by Crainic et al. (1997) have compared distributed synchronous and asynchronous cooperative search with sequential search. They found the solutions obtained to be superior. Furthermore, they conclude that, of the methods they reviewed, asynchronous cooperative search is the most effective. From the literature it is clear that cooperation can lead to better solutions. It is also robust, covering more of the search space and often leads to better performance, or speed-up (Alba, 2005).

2.3 Agent-based approaches in cooperative search

Crainic and Toulouse (2010) have produced a comprehensive survey and analysis of

parallel meta-heuristics and the classification schema of Crainic et al. (2005) explains clearly the diversity of the field. However in this survey they do not mention agent-based approaches. This is probably as much to do with the fact that the term is used to describe many different approaches. To this end, this schema will be used to help clarify the types of parallelism used by agent-based system and will help highlight any areas not covered by the schema.

In the literature, the term *agent*, is often used, as above, to describe distributed sub-processes. From now on the term multi-agent system (or agent-based system, the terms will be used interchangeably) is a system where the agents are autonomous programs communicating by message passing. A full definition of agents will be given in section 3.2, but for now the following will suffice.

Wooldridge (2009) defines an agent-based system as

> ... a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives.

Aydin (2007) has pointed out that multi-agent systems offer a natural way to implement cooperative search. Furthermore, he describes meta-heuristic agents as having the potential to carry out cooperative search where each agent implements either different or the same algorithms while exchanging useful information about the search between them.

In combinatorial optimization the term "agent" is often used to describe parallel population based heuristics such as parallel genetic algorithms (Vallada and Ruiz, 2009) or parallel ant colony heuristics such as Dorigo et al. (2006). The term has also been used by researchers referring to different parallel meta-heuristic approaches.

The Cooperative A-Teams of Talukdar et al. (2003) is an agent-based system that uses asynchronous agents which execute different solution modifying techniques where they can be classified in more than one way using the schema of Crainic et al. (2005). An agent will scan the current population from a central memory and select a

number of solutions, modify them and then write the population back to memory. An agent maybe a creator or a destroyer. A creator agent will augment the population of solutions when it modifies a solution while a destroyer will remove solutions from the population. A-teams also work on two levels. A complex problem can be broken down into sub-problems which an A-team agent will work on with its own memory pool for the sub-problem. From this description it can be seen that A-team agents are at once pC/C/MPDS when the creator and destroyer agents update the central pool. They can also be classed as pC/KC/MPDS when an agent has its own sub-pool.

This is due to the fact that an agent-based system according, to Wooldridge's definition will have a degree of *autonomy*. This implies that the agent will be *asynchronous* because it will have an internal communication mechanism; it will communicate *directly* and will also maintain its own internal partial view of the search environment. Therefore, by the classification schema it will be pC/KC or C/MPSS or MPDS meaning that there are multiple processes and no master, information is shared asynchronously and directly where the agent will have some kind of internal memory and agents can instantiate the same or different meta-heuristics. As a consequence they do no easily fall under the classification of Crainic et al. (2005). This is due to their *autonomy* according to their survey pC/C/X systems usually cooperate via a central shared memory while pC/KC/X will use adaptive-memory pools. Agent based systems communicate not through a shared memory (of any sort) but directly using protocols. This means that unless there is an agent acting as a central or adaptive pool, the agents must maintain an internal partial representation of the search.

Furthermore the agents in this thesis can conduct all of the problems tested as stand alone agents and reach a solution. They only cooperate because it improves solutions. What this means is that there is no governing process guiding the search and the agents are completely autonomous. The agents each participate in a search that amounts to a distributed meta-heuristic but the exact character of this heuristic can only be known at the end of the search (see chapters IV and V). It is therefore

proposed to call this type of algorithm an *emergent distributed algorithm*.

It is important to realise that an agent-based system is not just a collection of distributed processes or threads working together to solve a problem. Each agent in an agent-based system is an autonomous program in its own right. It can perform its allotted task without recourse to a governing process. This means that the multi-agent will have an internal representation of its environment and will respond through cooperation protocols to other agents accordingly. Another feature of such systems is that they communicate by passing messages as opposed to function calls to other processes. In a message passing system, the agent has to translate a message from its own internal representation into a text message that can be transmitted to another agent. A receiving agent must de-parse the message into its own internal representation.This means that an agent is not some kind of homunculus performing a task or subtask for some other controlling process. Agent systems are often, but not always, distributed over many machines or even the internet.

2.4 Conclusion

There is very little research into agent-based cooperative search. As a consequence, the classification mechanism of Crainic et al. (2005) has been used to survey current research into parallel meta-heuristics and what there is on agent-based cooperation. This has been undertaken to show that parallel meta-heuristic research is very different from agent-based research. Another aim was to show that there is very little research into agent-based cooperative search mechanisms using different meta-heuristics. It is proposed that this type of agent-based system can implement a cooperative search where each participant is an equal partner, but with the crucial property that they can complete a search on their own. They only cooperate because it is more beneficial, but it also means they cooperate using a type of algorithm called here *emergent distributed algorithm*. This idea will be expanded in chapter IV.

CHAPTER III
INTRODUCTION TO MULTI-AGENT SYSTEMS

3.1 Introduction

The aim of this chapter is to explain the rationale behind agent-based technology and to introduce the JADE platform on which the work of this PhD has been developed. Also, agents and agent-based systems are defined. In the first section 3.2 the question is posed what is an agent and what makes an agent-based system? A history of agents is provided in section 3.3. The FIPA standards (FIPA, 2000) for agent-based systems are introduced. Also the FIPA compliant JADE agent development platform (Bellifemine et al., 2007) is introduced in section 3.4.

3.2 What is an Agent?

There are many systems that called to be agent-based systems in the literature. Before giving the definition of an agent that will be used in this thesis, it is instructive to look some these other systems as a contrast with the one provided later in this section.

There are many types of agents described in the literature. Here are just a few:

- **Multi-agent.** A multi-agent is an agent that exists on an multi-agent system. This a term from Distributed Artificial Intelligence (DAI) where the multi-agents are a collection of problem solvers which can only solve a given problem by working together (Jennings et al., 1998).

- **Intelligent agent.** This is an agent that exhibits some for sort of Artificial Intelligence techniques such as learning (Woodridge and Jennings, 1995).

- **Software agent.** A software agent is a program that acts on behalf of a user (Nwana et al., 1999).

- **Autonomous Agent.** These agents have their own view of an environment and are capable of modifying the way in which they achieve their objective (Franklin and Graesser, 1997).

- **Distributed agent.** These are software entities that have the ability to communicate with other agents (Sycara et al., 1996).

- **Ant colony or Swarm agent.** These are examples of population or evolutionary heuristics which take their take single or groups of solutions, with or without parameter attributes, to be agents (Dorigo et al., 2006).

Given that there are so many uses of the word agent in artificial intelligence and operational research, it is important to explain what will be meant by an agent in this thesis. According to Ferber (1999) a software agent is a "computing entity which

a) is in an open computing system (assembly of applications, networks and heterogeneous systems),

b) can communicate with other agents,

c) is driven by a set of its own objectives,

d) possesses resources of its own,

e) has only a partial representation of the other agents,

g) possesses skill (services) which it can offer to other agents,

i) has behaviour tending towards attaining its objectives, taking into account the resources and skill available to it and depending on its representations and on the communications it receives."

Wooldridge, in the introduction to his book on multi-agent systems (Wooldridge, 2009) states that the

"history of computing to date has been marked by five important and continuing trends:

- ubiquity;
- interconnection;
- intelligence;
- delegation;
- human-orientation."

He argues that these strands, taken together, have lead to the emergence of a new field in computer science: *multi-agent systems*. He defines an agent as a computer system that is capable of *independent* action on behalf of its user or owner. In other words an agent can figure out for itself what it needs to do to perform its design objectives rather than having to be explicitly told at any given moment.

Given these definitions an agent will be defined in thesis as a software entity that is connected to an open communication system and is able to communicate with other agents. It is also autonomous and able to maintain its own internal representation of a computing environmental. Such an agent is also able to execute any number of behaviours that have been assigned to it. These behaviours are characterised by the appropriate interaction with the environment and other agents and possibly human users to achieve its goals.

Most modern multi-agent systems such as ZEUS (Nwana et al., 1999), FIPAOS (Poslad et al., 2000) and JADE (Bellifemine et al., 2007) adhere to the definition of multi-agent systems of Ferber (1999). "The term 'multi-agent system' (or MAS), is applied to a system comprising the following elements:

(1) An environment, E, that is, a space that generally has a volume.

(2) A set of objects, O. These objects are situated, that is to say, it is possible at a given moment to associate any object with a position in E. These objects are

passive, that is, they can be perceived, created, destroyed and modified by the agents.

(3) An assembly of agents, A, which are specific objects ($A \subseteq O$), representing the active entities of the system.

(4) An assembly of relations, R, which link objects(and thus agents) to each other.

(5) An assembly of operations, Op, making it possible for the agents of A to perceive, produce, consume, transform and manipulate objects from O.

(6) Operators with the task of representing the application of these operations and the reaction of the world to this attempt at modification which we shall call the laws of the universe (cf. Chapter 4)."

A short history of agent-based research is provided in the next section to put these definition into context with operational research and artificial intelligence.

3.3 A brief history of agent-based research

According to Wooldridge (2009) the notion of an Agent as some sort of computational artefact goes back to the early days of AI in the 1950's. It can clearly been see in Turing's famous test where a human sitting in front of a teletype terminal, after 5 mins of conversation, has to judge whether the "person" at the other end of the teletype terminal is indeed human or machine (Turing, 1950). Wooldridge argues if it is a machine at the other end of the teletype, then that machine must have a degree of autonomy which he argues is one of the basic traits of agent-hood.

However software agent research did not really get going until the 1990's. In the intervening years the seeds of agent technology were sown. A lot of research during that time was focussed on symbolic representation and their use in planning systems such as STRIPS (Fikes and Nilsson, 1972).

Another strand of research was conducted on concurrency and parallelism including Hewitt et al. (1973); Hewitt (1977) on actors and Gasser and Huhns (1989) on

Distributed Artificial Intelligence (DAI).

Agha (1985) proposed an *actor model* of concurrent computation for distributed systems based on the work of Hewitt et al. (1973). An actor is described as a computational entity that, in response to a message it receives, can concurrently:

- send a finite number of messages to other actors
- create a finite number of new actors
- designate the behaviour to be used for the next message it receives.

An actor may be viewed as an object augmented with its own control, a mailbox and a globally unique immutable name. They communicate asynchronously where no message to any of the actors is given priority over another. In this way the communication is deemed to be *fair*. Furthermore the path a message takes, and any network delays it may encounter, are not specified. Therefore the arrival and order of message is indeterminate. Other properties of actors systems include *encapsulation* of state and *atomic* execution of a behaviour in response to a message and *location transparency* enabling concurrent execution and actor mobility. These are all important ideas which laid the foundations of agent autonomy, communication and the fact that they are distributed.

Dissatisfaction with the symbolic AI of the 1960's and 70's le,d in the late 1980's, to the growth of behavioural AI (Brooks, 1990) which turned away from direct symbolic representations and looked to nature in order to focus on a more "holistic" view of systems as many simple processes collaborating to produce "emergent" system wide properties.

Also in 1970's and 1980's research into blackboard systems (Engelmore, 1988) and parallel blackboards (Carver and Lesser, 1994) showed that intelligent communication between processes could be achieved through shared data structures where relevant knowledge could be exchanged.

Along with blackboard systems, another technology that would be very influential in agent development even to this day, was being developed, that of the Contract

Net (Smith, 1980). The key concept was that a number agents solved problems by delegating sub-problems to other agents. It also introduced the notions of agent-based systems being economies that compete and negotiate and cooperate.

All these strands of research came together in the early 1990's and began to coalesce into modern agent research. Firstly, a number of projects funded by the European Union allowed researchers to work on agent-based systems that were autonomous, that could communicate relevant information focussed, but not exclusively, on behavioural AI. The MAGMA Demazeau and Müller (1990) was an example of such a project.

While in the United States the Defence Advanced Research Projects Agency (DARPA) funded the knowledge sharing effort INTERCHANGE (1998) which brought important advancements in agent communication including the Knowledge Query and Manipulation Language KQML (Finin et al., 1994) and Knowledge Interchange Formal KIF (Genesereth et al., 1992) and Ontologies (Gruber, 1993).

Also the growth of the world wide web was an important development for the growing agent-based research, allowing open world-wide communication through the internet protocol IP and the development of computer languages such as JAVA conceived to work with the internet. This enabled agent-based communications to be open and scalable in a way not possible before.

The growth of electronic commerce as result of the WEB lead to agent-based technology being designed for e-commerce. Early examples were FireFly developed by Shardanand and Maes (1995) and MAPPA Arafa et al. (2000).

At this time, there were moves to standardise agent-based systems and development. This resulted in the Foundation of Intelligent Physical Agents (FIPA) (FIPA, 2000), an IEEE body which produces standards for agent development and agent platforms. This was to enable agents developed on different FIPA compliant platforms to all perform full agent based communication with each other. A number of agent development platforms and environments have been developed as a result of this standard including ZEUS (Nwana et al., 1999). FIPAOS (Poslad et al., 2000)

and JADE (Bellifemine et al., 2007).

3.4 JADE

JADE (Bellifemine et al., 2007) is an open source FIPA compliant development platform. The framework described in this thesis is developed on JADE. This section will describe JADE and the FIPA compliant technologies it uses which are essential features of any agent-based system.

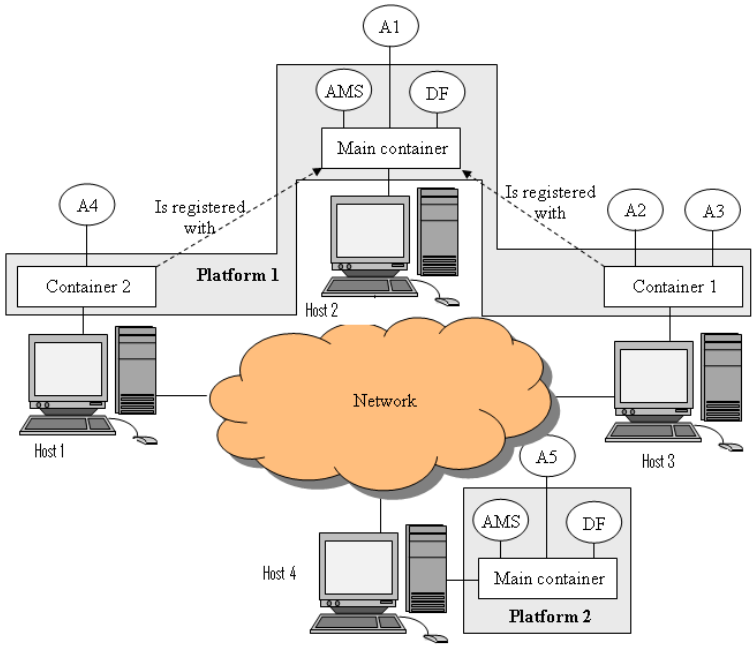


Figure 3.1: The JADE architecture

JADE is a FIPA compliant platform and as such, it must provide a number of important services (see diagram 3.1 Grimshaw (2011)). It must implement a number of Message Transport Protocols (MTP) so that the platform can communicate with other platforms. The most common is HTTP. The JADE environment comes with two important predefined agents:

a) the *Agent Management System* (AMS). This is the main authority for the platform and is the only agent that can create and kill agents and shut down the platform,

b) the *Directory Facilitator* (DF) which advertises the services of agents on the platform so that other agents can find them.

The platform also has a GUI facility and facilities to log and view the messages passed between agents. Developers can use the platform to build and test new agents that exhibit new behaviours and perform new tasks and services. This means the developer is only concerned with developing new agent behaviours and what they will communicate rather than with the problems of how to build agents.

Developing new agents requires a number of different techniques. To this end, JADE has a number of standard communication types implemented as behaviours or groups of behaviours. These standard communication types have been codified by FIPA into a number of protocols which a FIPA compliant platform must implement FIPA (2009). The FIPA list of protocols is by no means exhaustive, but contains ready-made communication protocols characterising common human communication activities. Agents communicate using FIPA-ACL FIPA (2000) which defines a number of conversation primitives deemed to be common to all human speech acts (Searle, 1970). The primitives, called by FIPA *performatives*, include *requesting, querying* and *informing*.

3.4.1 Agents and Behaviours

A JADE agent is a programme that schedules behaviours. When an agent is started it registers with the platform and therefore can be managed on the platform by the AMS and located by the DF. When other agents join the platform they can locate other agents through the AMS and DF.

A behaviour is a task that an agent executes. It is possible to schedule when and in what order these behaviours are to be executed. A behaviour will often involve an agent communicating with other agents and perform actions as a result of communication. JADE provides a number of predefined behaviours based on the most common tasks agents are required to perform. These behaviours include One-shot behaviours, Cyclic behaviours, Parallel behaviours and more complex behaviours. These are are

all arranged in a class hierarchy with the class Behaviour as the root class. It has two important subclasses CompositeBehaviour and SimpleBehaviour. The subclasses of CompositeBehaviour deal with ways a developer might want to build a complex behaviour with many sub-behaviours. While the sub-classes of SimpleBehaviour execute one task or behaviour at a time but some of these are specialised. For instance, WakerBehaviour which has a facility to sleep for specified periods and then wake-up to perform a task and then go back to sleep. When a behaviour terminates, control is passed back to the agent which then schedules the next behaviour or suspends the agent into a wait state or the agent is terminated by the AMS.

3.4.2 Agent Communication

Agent communication is probably the most fundamental feature of JADE and is implemented according to the FIPA standard. Agent communication is also an important feature of making an agent *autonomous* in that a sophisticated communication system is an important feature of maintaining an internal representation of an environment. JADE agents use asynchronous message passing where each agent has a mailbox where messages are sent by other agents. Whenever a message is posted into an agents mailbox message queue the receiving agent, is notified. However the agent chooses when it will act upon any messages received into its queue.

3.4.2.1 Performatives

Jade agents communicate using a language called FIPA Agent Communications Language FIPA-ACL. FIPA-ACL is developed from the speech acts theory of Searle (1970). The idea is that there are some basic communicative acts that are present in all speech and transcend the technicalities of different languages. For example if an agent REQUESTS something of another agent, the sender intends that the receiver performs some action. If an agent INFORMS another agent, then the sender wants the receiver to be aware of some fact. Finin et al. (1994) developed KQML as part of the DARPA knowledge sharing effort. However FIPA FIPA (2008) has standardised

the following performatives:

- accept-proposal- The action of accepting a previously submitted proposed to perform an action.
- agree- The action of agreeing to perform a requested action made by another agent. Agent will carry it out.
- cancel- Agent wants to cancel a previous request.
- cfp- Agent issues a call for proposals. It contains the actions to be carried out and any other terms of the agreement.
- confirm- The sender confirms to the receiver the truth of the content. The sender initially believed that the receiver was unsure about it.
- disconfirm- The sender confirms to the receiver the falsity of the content.
- failure- Tell the other agent that a previously requested action failed.
- inform- Tell another agent something. The sender must believe in the truth of the statement. Most used performative.
- inform-if- Used as content of request to ask another agent to confirm whether a statement is true or false.
- inform-ref- Like inform-if but asks for the value of the expression.
- not-understood- Sent when the agent did not understand the message.
- propagate- Asks another agent to forward this same propagate message to others.
- propose- Used as a response to a cfp. Agent proposes a deal.
- proxy- The sender wants the receiver to select target agents denoted by a given description and to send an embedded message to them.

- query-if- The action of asking another agent whether or not a given proposition is true.
- query-ref- The action of asking another agent for the object referred to by an referential expression.
- refuse- The action of refusing to perform a given action, and explaining the reason for the refusal.
- reject-proposal- The action of rejecting a proposal to perform some action during a negotiation.
- request- The sender requests the receiver to perform some action. This is usually to request the receiver to perform another communicative act.
- request-when- The sender wants the receiver to perform some action when some given proposition becomes true.
- request-whenever- The sender wants the receiver to perform some action as soon as some proposition becomes true and thereafter each time the proposition becomes true again.
- subscribe- The act of requesting a persistent intention to notify the sender of the value of a reference, and to notify again whenever the object identified by the reference changes.

Agents use FIPA-ACL as a way of communicating to a receiver the intention behind a message and to indicate if any action is required. Another important part of a message is letting the receiver know how to read the content of a message. This function is handled by an ontology which can be associated with any message.

3.4.3 Communication Protocols

It is important in agent messaging to develop messaging schemas for solving different problems. In the framework for combinatorial optimisation the agents use a

sequence of messages as part of a behaviour to achieve various actions. These message sequences are called communication protocols.

FIPA has specified a number of communication protocols and JADE implements them all as complex behaviours. Here is a list of the main protocols implemented by JADE:

- FIPA-Request
- FIPA-Query
- FIPA-Propose
- Iterated version of FIPA-Request
- Contract-Net
- FIPA-Subscribe

It is possible to build more complex protocols from these basic protocols. Indeed the framework described here uses a complex protocol built from Iterated version of FIPA request and Iterated Contract-Net (see Section 5.2 for more details).

3.4.3.1 Ontology

Ontologies play an important role within the agent communication. They define a set of general representational primitives with which to model a conversation will take place and as such are semantic (Gruber, 1993). JADE provides a comprehensive ontology representation system that make the process of implementing an ontology quite easy. Firstly it is important to design an ontology with respect to the problem trying to be solved and the communication required by the agents to solve this problem. Chapter V explains how combinatorial optimisation is modelled and how an ontology has been developed to express this model in this thesis. Once this is done all that is required is to implement Java classes of each of the primitives represented within the ontology. Each of these classes then must implement a JADE interface

which declares whether the primitive is an *action* or a *concept*. An *action* implies that the primitive is describing an action that is to be taken. A *concept* indicates that the primitive is about something. Finally the objects to be implemented are declared in an object that is a sub class of a BeanOntology object. JADE then seamlessly adds these to any message sent between agents. By the same token for any agent receiving messages, JADE will de-parse these messages and create the Java objects for the ontology.

This process enables the agents to communicate semantic quality in any message. In this thesis an ontology has been created that allows the agents to reason about a generic model of combinatorial optimisation. This raises the level of generality allowing only the agents to reason in these primitives about different combinatorial optimisation problems.

3.5 Conclusion

Agent technology has a long pedigree going back to the very inception of AI. The various strands of research that are the foundation of agent technology came together in the 1990's and since then the field has been growing. Agent technology was standardised in the late 1990's by FIPA but that does not mean that this is a closed field of research. Much new work is being done and it is a very rich field of research.

JADE is probably the most popular opens source development framework that came out of the work of the late 1990's offering a FIPA compliant platform that is constantly being updated. There is a rich developer network offering new add-ons to the basic platform such as the XML messaging add-on used in this thesis which means all messaging is done in XML. There are others for different ontologies and a version of the framework that can work on mobile phones.

CHAPTER IV
PROPOSED NEW FORMAL THEORY FOR COOPERATIVE ALGORITHMS

4.1 Introduction

It is the aim of this chapter to offer an explanation of cooperation by examining the various mathematical theories of interaction, actors, process algebras and the various proposed extensions to Turing Machines. It will be necessary to consider the mathematical and logical foundations of computation which have in the last 15 years come in for re-examination and criticism (Syropoulos, 2008). This, in turn, will lead to a discussion of *hypercomputation, interaction, the Church-Turing thesis* (Wegner, 1998; Goldin, 2000; Van Leeuwen and Wiedermann, 2001) and *unbounded non-determinism* (Agha and Hewitt, 1987).

For these reasons, this chapter is set out in two halves sections 4.2 and 4.3. The first half, section 4.2, consists of a review of the literature it introduces Turing computability and criticisms of as a theory of modern computing. In subsection 4.2.1 the relevant literature on interaction and agents in computation theory, computability and hypercomputation is reviewed. In subsection 4.2.2 systems of cooperating agents are examined.

In section 4.3 cooperating algorithms are introduced, and a new refinement of Turing machines is proposed. The implications of this new refinement are discussed and a refutation to claims that interaction defeats the Church-Turing thesis is also proposed. While subsection 4.2 proposes a theory of cooperation which refutes the view, held by many reviewed in the first section, that there are counterexamples that mean the Church-Turing these is false.

Finally, in section 4.4, conclusions and suggestions for future work are offered.

4.2 Literature Review: Turing computability and modern computing

The Turing model of computation (also known as computability) has been criticised as not being adequate to explain and describe modern computer systems. Traditionally a computation is considered successful, if after a certain amount of time, it halts and gives an output. Therefore Turing machines are defined as being able to compute partial or total functions. They are also defined as stand alone devices where input is read from, and output is written to, a tape supplied before a computation starts.

Contrast this with modern computers where operating systems or web-servers are designed never to stop. They do produce outputs, but they do not stop to produce their output as a traditional Turing machine would. They are also connected together through local area networks and the wider internet. To this end they receive inputs from many different locations, often as background processes, so that the user is unaware of their existence. Also a user can interact with a modern computer supplying input in response to the computer's output in a sort of feedback loop. In this context, it is argued, it is hard to see how they compute partial or total functions. Furthermore, modern machines are not Turing machines and a new model is required.

The Church-Turing thesis especially, has come under much criticism. Goldin and Wegner (2005) in particular, have offered counterexamples to the thesis proposing Interaction and Persistent machines to account for the operation of modern computers which are not classical Turing machines. The hypercomputation community (Ord, 2006; Stannett, 2003) has also criticised the Church-Turing providing other physics inspired counterexamples (Syropoulos, 2008). While Agha and Hewitt (1987); Milner (1982) have not sought to challenge the logical foundations of computing, they have offered models of the process of computing, to account for the way modern machines interact with people, other machines, and their environments.

Turing derived his model of computing in order to show there was no decision process for every statement of first order logic. In doing this his project was to find a systematic mathematical definition for the informal abstract notion of an algorithm.

He devised a thought experiment where Turing machines are an idealised computing device that can instantiate our informal understanding of algorithms and, in so doing, make them precise so that they can be treated mathematically. Since then, Turing's work, taken along with the work of Church (Church, 1932), Kleene (Kleene, 1952) and many others, has been developed into a logical mathematical theory of computation (computability) on which all computers are supposed to be based.

In this chapter the practice of deriving counterexamples to the Church-Turing thesis from the features of modern computers is examined together with the practice of offering *extensions* to Turing machines, based on these features. This normally involves a conflation between the physical/causal interactions of a physical computer and the abstract informal definition of an algorithm. For an argument from the physical to the abstract to work, a definition of the causal interactions of the physical process needs to be given for this definition to be meaningful.

One such property, which has come under much discussion lately is *unbounded non-determinism* which arises naturally in the context of computer interaction. It is hard to see how the notion of abstract algorithm can be modified to account for this and other physical phenomena. However, if computers do not implement abstract algorithms, they implement something very similar to them. Therefore, the question arises what is the correct level of description and what are the consequences for traditional computability?

In this chapter these issues are discussed with respect to agents which, it is argued, can exhibit the physical property of *unbounded non-determinism*. By producing a refinement of a Turing machine, it is possible to show that, taken as a process algebra, this refinement accounts for unbounded non-determinism. If however, it is taken as an abstract theory, then a clarification of Turing theory is proposed.

4.2.1 Interaction machines, Persistent Turing machines, Site Machines, Actors and process algebras.

There is no literature offering a formal description of cooperating agents in the standard theory of computation. However, a number of researchers have offered formal extensions to the theory of computation which exhibit physical or algorithmic properties that agents share. In this section these different extensions are examined and their relationship with cooperating agents is established.

4.2.1.1 Interaction, Persistent and Site Machines

Wegner (1998) defines his Interaction Machines (IM) as machines that are able to dynamically read and write streams that enable it to interact with the outside world. What makes IM's different is their ability to communicate with an environment synchronously or asynchronously. The streams are temporal character streams. IM's are also open in the sense that it is possible to understand their working by studying their *stream histories*. Streams are an interactive time-sensitive analogue of strings. He argues that because of these properties it is easy to see that IM's cannot be represented as Turing machines because IM's are more expressive.

Wegner argues that because these machines can send and receive streams, they are more powerful than Turing machines. He formally describes streams containing characters of an Interactive Grammar (IG). An Interactive Grammar is defined as a grammar $IG = (N, T, S, R)$ which accepts streams by reduction rules R, i.e. with a dynamic listening operator "." and a *non-deterministic choice operator* "+". So, for example, the grammar for streams: $stream \longrightarrow (+1).stream$ accepts binary streams by the "." operator *listening* for externally controlled input , while the "+" operator expresses incremental choice and commitment (Wegner, 1998). Wegner's point is that the next step in computation of an IM is externally controlled. This is not Turing computable. He goes on to extend this by describing an Interactive machine with an interactive grammar M1. If another machine M2 is defined with all the past histories of M1 then it is more powerful than M1.

Goldin (2000) has proposed Persistent Turing Machines which she also claims take computing beyond the "Church-Turing barrier".

> *Persistent Turing Machines* (PTM) are multitape machines with a persistent worktape preserved between interactions, whose inputs and outputs are dynamically generated streams of tokens (strings). They are a minimal extension of Turing Machines (TMs) that express interactive behaviour.

A persistent Turing machine transforms an input stream to an output using a function. In the course of its operation the state of the machine changes. The evolving state of the machine is recorded on its work tape and the interaction can be seen as a stream of pairs. The left-hand element in the pair is from the input stream and the right-hand from the output stream.

She recognises that this type of machine only captures a limited form of interaction which she calls *sequential interaction* (Goldin, 2000).

> "Sequential interactive computation continuously interacts with its environment by alternately accepting an input string and computing a corresponding output string. Each output-string computation may be both non-deterministic and history-dependent, with the resultant output string depending not only on the current input string, but also input strings" (Goldin and Wegner, 2005)(section 6.2).

Van Leeuwen and Wiedermann (2001) introduced Site and Internet machines to model individual machines which might be connected to others in a network of site machines (Van Leeuwen and Wiedermann, 2001). A site machine is modelled on a personal computer: specifically a personal computer that has a hard disk and can communicate with its environment by sending and receiving streams of data through a number of input and output ports.

The hardware and software of a site machine can be changed by an an external operator which is part of the machine's environment and is able to upgrade the machine. If a function γ is defined to return a description of the hardware/software upgrade, it will return an empty string if no upgrade takes place, otherwise it returns the upgrade description. Van Leeuwen and Wiedermann (2001) argue that this function is not computable in the traditional sense because it takes place at a specific moment in time and its values cannot be defined in advance.

To this end, a site machine performs a computation that performs an incremental symbol by symbol translation of an infinite multiple stream of input symbols to a similar stream of output symbols. More formally, if the site has k input ports and l output ports, where $k, l > 0$, then the site machine computes a mapping Φ of the form $(\Sigma^k)^\infty \longrightarrow (\Sigma^l)^\infty$ where Σ is a finite alphabet. In this way an infinite stream of k-tuples is incrementally translated into an infinite stream l-tuples.

These new machines exhibit three new features: *advice, interaction* and *infinity of operation.* In the definition of site machines external information can be presented the machine through the use of oracles. However, Van Leeuwen and Wiedermann formally define them as advice functions. Site machines start initially with tapes filled with blanks. The machine's operation depends on some controlling device. At each step the machine reads symbols which appear at its input ports and writes symbols to its output ports. What the machine will do next is dependent on what it has already read, what lies under its scanning heads and the instructions being executed. It can also at any time t_i consult its advice but only for values $t_i \leq t_n$ where $i, n \in \mathbb{N}$.

4.2.1.2 Actors and process algebras

Agha (1985) proposed an *actor model* of concurrent computation for distributed systems based on the work of Hewitt et al. (1973). An actor is a computational entity that, in response to a message it receives, can concurrently:

- send a finite number of messages to other actors

- create a finite number of new actors
- designate the behaviour to be used for the next message it receives.

An actor may be viewed as an object augmented with its own control, a mailbox and a globally unique immutable name. They communicate asynchronously where no message to any of the actors is given priority over another. In this way the communication is deemed to be *fair*. Furthermore the path a message takes, and any network delays it may encounter, are not specified. Therefore the arrival and order of message is indeterminate. Other properties of actors systems include *encapsulation* of state and *atomic* execution of a behaviour in response to a message and *location transparency* enabling concurrent execution and actor mobility. Agha (1985) characterises actors as processes. This characterisation has enabled the ready comparison of this model in process models such as the π-calculus of Milner (1982).

Milner (1982) sought to extend the Lambda-Calculus of Church to account for the properties observed in modern computing. He generalised these properties modelling an entity as a process which interacts with other processes. To this end it is a mathematical theory that can describe the functionality of computer models or systems.

subsubsectionUnbounded non-determinism Both the actors theory and the π-calculus each account for the physical phenomenon of unbounded non-determinism. Actors have the special properties of autonomy of action, of messaging, of being able to co-operate with each other and being able to move freely from one environment to another. The Actor model is finding application within multi-agents systems although actors are considered to be more general than agents because agent systems tend to impose extra constraints upon actors. These include the requirement that agents have commitments and goals. However, in this chapter, the discussion of agents can apply equally to actors.

Carl Hewitt (Hewitt, 2006a) in his participatory semantics of *actors* suggests that:

> Actors are universal primitives of concurrent computation. In response

> to a message it receives, an actor can make decisions, create more Actors send more messages, and designate how to respond to the next message received. (Hewitt, 2006a)(pp2-3)

Hewitt's work has been often cited in agent theory (Wooldridge and Jennings, 1995)(pp4). One of the crucial concepts Hewitt introduced is the property exhibited by concurrent systems of *unbounded non-determinism*.

> *Unbounded non-determinism* is the property that the amount of delay in servicing a request can become unbounded as a result of arbitration of contention for shared resources *while still guaranteeing that the request will eventually be serviced.* (Hewitt, 2006b)(p1)

If agents communicate over contentious resources while guaranteeing message delivery then the order in which messages will arrive at the agent will be unpredictable. Therefore, it is clear that it is no longer possible to predict the future behaviour of an agent.

Consider the example of computer systems communicating over the internet from different sides of the globe. This interaction will be subject to unpredictable delays due to how the internet works. If the computer systems are sufficiently autonomous and are communicating asynchronously such as agents, it is clear that *unbounded non-determinism* would cause the agents to behave unpredictably as different messages arriving at different times would cause different behaviours in the agents.

Unbounded non-determinism does not occur just over the internet, but over any concurrent communication medium where resources are guaranteed but are limited. For example, if humans cooperate in a joint venture using the postal system (assuming that delivery is guaranteed) as their means of communication then unbounded determinism will occur. It is not a property of computers alone.

4.2.1.3 Arguments from physical phenomena against the Church-Turing Thesis

Criticism to these arguments has been made by both Cotogno (2003) and Rocha (1995) suggesting that it is a circular argument to extend the Church-Turing thesis by referring to observations of physical computing devices. Turing's great insight was to offer a mathematical description of informal *mathematical effective procedures*. In other words, to argue from the physical to the mathematics is somewhat circular. As Rocha puts it, the thesis offers a sort of *mathematical alibi* by which mathematicians can write an informal algorithm for some mathematical function, and by invoking the Church-Turing thesis, save themselves the necessity of writing out in longhand the full recursive function that computes this effective procedure. What Wegner, Goldin, van Leeuren, Wiedermann and others are doing is arguing against the Church-Turing thesis by using descriptions of physical systems to represent *mathematical effective procedures*. As Rocha (Rocha, 1995) (pp. 65) remarks:

> "There is a circular argument in all the extensions of the Church-Turing Thesis beyond mathematics; the very first assumption that physically realizable dynamics represent effective procedures is just what the thesis aims at hypothesizing."

If computability is to be refuted, what is needed is a full mathematical theory of physical computation, rather than a few extensions to Turing machines to account for perceived differences. Cotogno in a similar vein argues that it is always possible to rewrite interaction machines' tapes into recursive functions by rewriting suitable sub-routines for all the interactions. Interaction therefore, in his opinion, is nothing more than an efficient computation method for computing certain functions.

However, Rocha points out that defining a new mathematical theory of physical processes is no easy matter. This is because if, from observation, it is possible to predict the outcome of a physical process,then there must be an identifiable causal sequence. If such a sequence exists then anything we can identify in this way is also computable (Rocha, 1995). For such theory to be successful, it is necessary to

identify which causal process are computable and which are not. Cotogno argues that the hypercomputation community, Wegner, Goldin, Van Leeuren and others, do not do this. Otherwise any putative counter-example device offered must be Turing computable.

> "The Church-Turing Thesis is thus used to correlate the computing abilities of formal devices with the causal dependencies of natural processes: if we can causally reproduce some natural process (extended effectiveness) then it is computable and it is performing an embodied computation. As a corollary, if we believe the universe is causal, then it is computable."

Both Rocha and Cotogno are uneasy with arguments from the physical to the abstract. They are worried that it is not possible to find the right level of description where the model is abstract enough to serve as a formal model without losing the descriptive power of the causal process. They are worried any such abstraction will be too weak and therefore will describe anything and nothing. As a consequence, they argue that the physical version of the Church-Turing thesis is just the same as the logical-mathematical definition.

The phenomenon of unbounded non-determinism is often cited by all as the main example of how interactive or agent-based systems are fundamentally different from Turing machines. Wegner and Goldin however, provide a formal Turing-like definition which they claim enables them to provide an extension to Turing machines to account for this physical process. From this they argue they have found a counterexample to the Church-Turing thesis. But unbounded-non-determinism and indeed Wegner's temporal input strings for IM's are physical phenomena which need causal explanation. In contrast, Milner and Hewitt do not shy away from this as their project is to offer theory of computation as computing.

It is questionable that they can account for the delay of messages due to the physical characteristics of a communication medium through a Turing-like definition.

Wegner, Goldin, Van Leeuwen and Wiedermann do not explain how these physical phenomena can be accounted for in their extensions of Turing machines.

However, it is not unprecedented to offer extensions to Turing machines. Turing himself, did so by proposing Oracle machines (Turing, 1939). However, these are logical extensions and, indeed, as has been argued, any extension has to be a logical one rather than a physical extension. It will be argued further that communication and cooperation are logical properties of effective procedures. Furthermore this property is not predictable and therefore not deterministic and therefore is an extension like an Oracle or a Choice machine. In section 4.3 this will be explored in more detail and the conclusion will be drawn that, while it will always be possible to refine Church-Turing thesis further by adding oracle machines or cooperating algorithms, this does not change the basic Church-Turing thesis or any other part of computability theory. It is unlikely that a true counter-example to the Church-Turing thesis will be found, unless a major new mathematical theory is proposed that changes current understanding.

4.2.2 Cooperating Agents

As explained in chapters III and II, agents are autonomous, they can work independently or they can also work with other agents to solve an internal or collective goal. They often work for an owner or user cooperating or negotiating with other agents on their behalf. In this context they have been used in shopping environments (Arafa et al., 2000) maintaining profiles of their owner's preferences in order to suggest relevant potential purchases. They have also been used in telecoms, healthcare and many other industrial applications (Bellifemine et al., 2008). Agent development platforms such as JADE have been released that (Bellifemine et al., 2008) enable developers to build agents with respect to the international standard of the Foundation for Intelligent Physical Agents (FIPA) (Dale and Mamdani, 2001).

When agents cooperate they can use these properties to completely change the outcome of a computation. Furthermore this can be completely non-deterministic

if more than two agents are participating, and they are communicating in an environment where message delivery is guaranteed but where resources are limited; in other words, unbounded non-determinism. This happens when, because of possible communication delays, it is not always possible to determine the order in which behaviour-changing messages are received. For example, if one agent asks some other agents to each provide information to help solve a problem, and each agent is tackling the problem differently, then, because of the uncertainty in delivery order of messages, the result of the first agents computation cannot be determined. This is because it is not possible to know which answer will be used by the agent to solve a problem. This is further complicated if some of the requested agents also need information from other agents.

Cooperation in agents is achieved by the use of cooperation protocols which are, in effect, distributed algorithms executed by participating agents where are each agent plays its part in the algorithm. But the outcome of the algorithm is not predetermined and the overall state of the algorithm is given by the collective states of all the agents at a given step in its execution.

In the context of this chapter a *state* (Hopcroft et al., 1979) is a particular set of instructions which will be executed in response to the agent's input. Input in this case just means data available to the agent either by communication or from some kind of storage. The combination of the input and state will determine the agents behaviour. Given these properties the following definitions are proposed.

Definition 1: An agent is capable of communicating its entire state and data.

Definition 2: Cooperation is the process of autonomous agents working together to the same end. This means agents participate in a distributed algorithm where no one agent controls the process and they all work towards a collective goal.

It can be seen that cooperating agents are subject to unbounded non-determinism 4.2.1.2. If this behaviour explained either by Agha (1985), or by process algebra

Milner (1982) where the physical conditions of this phenomena are accounted for, then it can be seen that agents behaviour cannot be predetermined. As Cotogno has argued it is possible to look at the agent system after the computation is completed and derive a Turing machine that will compute what the agents have done. Cotogno argues that the agents method of computation is just more efficient than a Turing machine. However, it would be good if it was possible to have a logical-mathematical definition that captured this efficiency. In the section 4.3 a more formal definition of cooperation will be provided where the definition of a Turing machine will be refined to account for cooperating algorithms.

4.3 A Proposed Turing machine definition of cooperation

Cooperation has been explained in section 4.2.2 as autonomous agents working together towards a collective goal. As a consequence, agents can exchange information that can change each others behaviour. Furthermore, it has already been explained that this behaviour is subject to the phenomenon of *unbounded non-determinism* where an agent's future behaviour a priori cannot be determined as a result of cooperation.

For what follows it is not necessary that an agent be a software agent or computing device. The above discussions about Software agents, interaction machines and the computational power of modern computers were given to show the following that:

a) the mathematical and logical foundations of computer science are being questioned by researchers

b) agents because of unbounded non-determinism are able to execute computations which stand alone computers cannot.

The following is purely a logical-mathematical discussion about cooperating algorithms or effective procedures and therefore does not necessarily depend on computation devices in the real world. It will show that cooperating algorithms are able to perform computations that are not possible for stand alone algorithms. It will

be argued that cooperation is indeed a logical property of effective procedures (or algorithms) and that this can be modelled by Turing machines with additional move operators. Informally, it can be seen that a Turing machine can communicate by use of a messaging tape which each machine shares and can read and write information. This means their instruction set will have to be augmented with the instructions to read and write to this tape. In this way algorithms can communicate and pass information between each other. This messaging tape can be infinite just as the normal internal tape of a Turing machine is also infinite.

When a cooperating Turing machine has an instruction to look into the messaging tape, it reads all the information on the tape, moving (without loss of generality) from left to right, and tries to find information that will enable it to further its ability to complete its tasks and form a conclusion. By the same token, it will be able to place information into the tape by the use of a *write* instruction in its instruction set.

On the face of it this is very similar to the storage work tape proposed by Goldin Goldin (2000). However her tape is used for the recording of past histories. What is proposed here is merely a way for cooperating algorithms to pass information to each other.

Recall a cooperating algorithm is part of a distributed algorithm that is not directly controlled by any other algorithm. There are two types of distributed algorithm available in the context of cooperation. These are:

- The computation of this distributed algorithm will halt only if each algorithm taking part halts. If any participant does not halt then the algorithm does not halt.

- Each participant in the distributed algorithm does not depend any other participating algorithm and will halt (if it is going to) irrespective of the other algorithms. However information from the other algorithms may well improve results.

Before going into further detail about cooperating algorithms and their Turing

machine representation, it is necessary to provide a formal description of a non-deterministic Turing machine.

4.3.0.1 Definition of a Non-deterministic Turing machine NTM

Firstly, a non-deterministic Turing machine is formally defined using the set theoretic model of Hopcroft et al. (1979). From this definition it should be easy to see how it differs from the cooperating version of a Turing machine described in the next section.

A non-deterministic Turing machine M can be formally defined as a 6-tuple $M = (Q, \Sigma, \iota, \beta, \delta, A)$, where:

Q = the set of finite states

Σ = the finite set of symbols used by the machine

$\beta \in \Sigma$ the blank symbol

$\iota \in Q$ the initial state of the machine

$\delta \subseteq (Q \setminus A \times \Sigma) \times ((Q \times \Sigma) \times \{L, R\})$ is a relation on states and symbols called the *transition relation*. $\{L, R\}$ are tape head moves. L means move one cell to the left. R means move one cell to the right.

$A \subseteq Q$ the set of accepting states

The NTM is different from a Deterministic Turing Machine (DTM) in that there can be more than one state to which the machine can legally move. In this respect the transition relation δ is different from the transition function of a DTM. This means a tree like structure can be drawn showing the state transitions of an NTM. The machine will try different next states branching in a tree like manner but will only move to accepting states.

It can be seen NTM has the property of *bounded non-determinism*. This means if an NTM always halts on a given input tape T, then it halts in a bounded number of steps, and therefore can only have a bounded number of possible configurations.

4.3.0.2 Proposed definition of a cooperating algorithm as a Turing machine (CNTM)

In this subsection a Turing machine definition of cooperating algorithm is proposed. Let a cooperating algorithm (CA) be an effective procedure that can participate with other similar algorithms in a distributed effective procedure. A distributed effective procedure as just described will be called a *conversation*. A CA is able by some method of communication to change the behaviour of another CA.

A cooperating Turing machine (CNTM) is defined to be a machine that models our (human) intuitive understanding of CA. It has a number of important distinguishing features:

- autonomy of action
- send and receive operators
- pursuing a collective goal

A CNTM is autonomous in the sense of the second type of distributed algorithm as described in section 4.3 in that it can complete its goal with or without sending and receiving information from the information communication tape. This means if there is no relevant information on the tape when it is directed to look, it will carry on with its calculation regardless. It is pursuing a goal or algorithm where it can use cooperation to greatly improve its chances of achieving a good result. The following discussion will focus on the second type of distributed algorithm as it clearly shows the differences between non-cooperating and cooperating algorithms.

A group of CNTM's participate in a distributed algorithm.

Let T be the set of all cooperating algorithms (CNTM) participating in a specific conversation.

Let Q be the set of possible states used by all CNTM in T.

Let Σ be the set of all possible symbols used by all possible CNTM's that could participate in a specific conversation. Let $M \subseteq Q \times \Sigma$ be the infinite unordered set of

all events passed as messages between the CNTM's during the conversation. These events will contain state and data information used by the CA's to alter each other's behaviour during the execution of the conversation.

A cooperating algorithm $T_i \subset T$ is a Turing machine which can be formally defined as an octuple $T_i = (Q_j, \Sigma_k, \iota, \beta, \xi_l, \varnothing, \delta, A_m)$ where $i, j, k, l, m \in \mathbb{N}$, and:

Q_j where $Q_j = Q_1 \cup Q_2 \cup ... \cup Q_j \subseteq Q$ the set of states.

Σ_k where $\Sigma_k = \Sigma_1 \cup \Sigma_2 \cup ... \cup \Sigma_k \subseteq \Sigma$ the set of symbols used by the CNTM.

$\beta \in \Sigma$ the blank symbol

$\iota \in Q_j$ the initial state of the CNTM

$\xi_l = (Q_a \times \Sigma_b) \subseteq M$ where $a, b \in \mathbb{N}$ is the event relation which is a pair consisting of state(s) and a symbol(s) from the CNTM's alphabet. These pairs are the messages passed in the conversation and are elements of the set of all messages M.

$\varnothing$ is the empty state where $\varnothing \in Q$

$\delta \subseteq (\xi_c \times \xi_d) \times \{L, R, S, RC\}$ where $c, d \in \mathbb{N}$. $\delta \subseteq Q_e \times \Sigma_f \times Q_g \times \Sigma_h \times \{L, R, S, RC\}$ where $e, f, g, h \in \mathbb{N}$ and $Q_e \times \Sigma_f \times Q_g \times \Sigma_h \subseteq Q \times \Sigma \times Q \times \Sigma$ is a relation on states and symbols called the *choice transition relation* which by the axiom of choice enables δ to choose the octuples necessary for the agent to move from state to state in a conversation between CNTMs. $\{L, R, S, RC\}$ are tape head move operators. L means move one cell to the left R means move one cell to the right. S is the send operator meaning copy the contents of the current cell to the next blank cell on the worktape M means read the next symbol on the worktape M.

$A_m \subseteq Q_i$ the set of accepting states

The CNTM operates much the same way as a standard non-deterministic Turing machine. It has a tape segmented into squares which can contain characters from the CNTM's alphabet. It has a scanning head able to read the contents of each square one at a time. It also has a means to print onto the tape. The CNTMs can also translate left or right along the tape or it can send or receive information to and from other CNTMs. There are a number of ways to describe how the send or receive operators might work in Turing-like thought experiment. For example, the CNTMs could be arranged so that their tapes are parallel to each other. The heads of each machine could not only move from left to right along the tape as in the original thought experiment but also from tape to tape to where $S \in \{L, R, S, RC\}$ writes symbols to another machine's tape and $RC \in \{L, R, S, RC\}$ reads symbols from another machine's tape. Another way of doing this, adopted here, is to define the unordered set M as the set where the read and write operators S and RC of each CNTM add and remove information by the use of the choice transition relation δ as described above. The way to think of this is as an unbounded infinite tape where the CNTMs can read and write information as required. To this end the tape does not contain the state of the conversation, just the information passed between each CNTM.

The transition relation defines a quintuple: $q_i\, S_j\, S_k\, \{L, R, S, RC\}\, q_l$ where $q_i, q_l \in Q_m$ and $S_j, S_k \in \Sigma_n$ and $\{L, R, S, RC\}$ means move left or right or send or receive. The send and receive move operators S, RC allow the CNTM to send states $q_i \in Q_j$ and data $S_k \in \Sigma_l$ to other CNTMs. CNTMs can receive state and data information by the RC move operator. It receives some state from some other set of states and some data from some other alphabet not part of the agent $q_p, S_q \in (Q_r \times \Sigma_s) \subseteq M$ and adds them to its own set of states and alphabet. This relation works much the same way as the transition function of an NTM. The CNTM's receive operator chooses from the set M the state and data pairs from this set that will lead the CNTM to an accepting state just as with an NTM. However, the set M is infinite and, as other CNTMs are sending state and data pairs to the set M, it is not clear in which order a

CNTM will receive new information. This means the choice transition relation does not guarantee the order in which a CNTM will receive new information that could change its behaviour and therefore it is not possible to know in advance the behaviour of a CNTM.

It is proposed that a CNTM *without* cooperation is the same as a NTM In this section it is proposed that a CNTM *without* cooperation is the same as a NTM. To operate as a normal non-deterministic Turing machine, it is necessary to configure the CNTM so that it does not have the ability to send or receive information to another CNTM. This means the CNTM can only move left or right along its internal tape. In effect only symbols from the agents alphabet will be written and read to and from tape when operating in this way.

Here is a simple example of this type of operation:

$$q_1\,0\,1\,R\,q_2 \quad q_2\,0\,1\,R\,q_1 \quad q_2\,0\,1\,R\,q_h$$

Here without loss of generality, assume that the CNTM only has the symbols 0 and 1 in its alphabet and that blank will be taken to be synonymous with 0. The tape initially is assumed to be blank. This CNTM then only has three states $q_1,\ q_2,\ q_h$. State h is the halting state where the machine stops. It starts in state 1 if it reads a 0 then it will print a 1 in that square and move one square to the right. However it has a choice for state 2 it can read a 0 print 1 and move to right and go back to state 1 in which case the machine never stops. It could choose the other version of state 2 in which case it reads 0, prints 1 moves to the right and then transitions to the third state where it stops. It can be seen that the agent can act like a normal non-deterministic Turing machine. Here the machine will opt to transition to state h as it only transitions to accepting states. State 2 is not an accepting state.

Theorem 4.1. *It is propsed that a CNTM is the same as an Non-deterministic Turing machine if and only if it does not communicate.*

Proof. Refer to the formal definitions of a CNTM and non-deterministic machines above. A non-deterministic machine M is a sextuple $M = (Q, \Sigma, \iota, \beta, \delta, A)$ an CNTM

as 8-tuple $T_i = (Q_j, \Sigma_k, \iota, \beta, \xi_l, \varnothing, \delta, A_m)$. If A does not communicate then it only reads data to and from the tape and only moves left and right along the tape. This means that the transition relation for the agent will be $\delta \subseteq (Q_i \setminus A_n \times \Sigma_j) \times ((Q_k \times \Sigma_l) \times \{L, R, \varnothing, \varnothing\})$. This means a legal configuration for this type of CNTM is defined as a quintuple:

$$q_i\, S_j\, S_k\, \{L, R, \varnothing, \varnothing\}\, q_l \equiv q_i\, S_j\, S_k\, \{L, R, \}\, q_l.$$

Hence a CA that does not communicate is an NTM. □

It is proposed that a CNTM *with* cooperation is *not* the same as a NTM In this section it is claimed that a CNTM *with* cooperation is *not* the same as a NTM. The difference between an NTM and an CNTM is that the latter can cooperate with other CNTMs by being able to participate in a distributed algorithm. This is because CNTMs can drastically change the outcome of a each others computations. Furthermore CNTMs participating in a distributed algorithm may well have different roles, which means they will be sharing information asynchronously. Therefore through the choice transition relation, the order in which the CNTMs participating in a conversation will send and receive information from the event set M, is not guaranteed. This is due to the fact the CNTM will read all the events in the set M and determine if an event will lead to an accepting state for the CNTM. It will do this in a tree like structure each time trying to find the best accepting state. But these tree like structures cannot be determined as it will be unknown which accepting states will be used by the algorithm. In this way the future behaviour of a CNTM cannot be predicted.

Here is an example of cooperating CNTM's. Imagine three CNTM's that are able to cooperate with each other. The first CNTM called A has the behaviour of reading a blank square on the tape and then replacing it with the letter A . The next two CNTM's B and C have the same behaviour but write the letters B and C respectively.

In this example the CNTM's cooperate by broadcasting their ability to print their respective letters. So if CNTM A broadcasts and B receives then B's behaviour will

be modified so that it will now print A. Similarly if C receives B's behaviour then C will print B. But because of the *choice transition relation* there is no guarantee which CNTM will receive A's broadcast first, and this is true for the others as well. Therefore this could mean at some time in the future there will be a CNTM A printing A,B or C and similarly for the other agents. Of course as time goes on all CNTMs will receive each others broadcast *But when this will happen and in what order cannot be predetermined* .

CNTM A

$$\mathrm{q}_{A1}\,0\,A\,R\,q_{A2} \quad q_{A2}\,0\,A\,R\,q_{A1} \quad q_{A2}\,0\,A\,S\,q_{A1} \quad q_{A2}\,0\,A\,R\,q_{A3}$$

$$\mathrm{q}_{A3}\,0\,A\,R\,q_{halt} \quad q_{A2}\,0\,X\,RC\,q_x \quad q_x\,0\,X\,R\,q_{halt}$$

CNTM B

$$\mathrm{q}_{B1}\,0\,B\,R\,q_{B2} \quad q_{B2}\,0\,B\,R\,q_{B1} \quad q_{B2}\,0\,B\,S\,q_{B1} \quad q_{B2}\,0\,B\,R\,q_{B3}$$

$$\mathrm{q}_{B3}\,0\,B\,R\,q_{halt} \quad q_{B2}\,0\,X\,RC\,q_x \quad q_x\,0,X\,R\,q_{halt}$$

CNTM C

$$\mathrm{q}_{C1}\,0\,C\,R\,q_{C2} \quad q_{C2}\,0\,C\,R\,q_{C1} \quad q_{C2}\,0\,C\,S\,q_{C1} \quad q_{C2}\,0\,C\,R\,q_{C3}$$

$$\mathrm{q}_{C3}\,0\,C\,R\,q_{halt} \quad q_{C2}\,0\,X\,RC\,q_x \quad q_x\,0\,X\,R\,q_{halt}$$

Here are the machine configurations for the above scenario 4.3.0.2. We take 0 to be synonymous with a blank cell. Here each CNTM has the same basic set of configurations. However each CNTM has a different initial alphabet of symbols it can read and write. CNTM A knows 0 and the letter A. Similarly CA B only knows 0 and the letter B and CNTM C 0 and C. Each CNTM starts in its state 1 (A1,B1,C1) reads 0 in a cell and overwrites it with the letter from its alphabet and then moves to the right. It then transitions to state 2. Here again it reads 0 and prints its letter and moves to the right but goes back into state 1. If it only had these two configurations each CNTM would be in a continuous loop. However just as in NTM's the CNTM can have more than one version of a state, one leading eventually to an accepting state. In the case of these three CNTM they also have three other state 2's. One state 2 is where the CNTM reads a 0 and prints its letter and moves to state 3. From

state 3 the CNTM reads a 0 moves to the right and then halts. Another state 2 sends its state (state 2) and its data (read 0)$q_{A2}\,0\,A\,S\,q_{A1}$ and then moves back to state 1. In this scenario, when sending information, the CA's are taken to be broadcasting to the other two. The final state 2 is for receiving a state and data from another agent. Here the state and data are not known a priori so the CNTM just has a variable X for the data received and a state x for the state received. Each CNTM has a final configuration where having received data X and state x, they move to that state and read 0 but print whatever X is and then halt. Figure 4.1 below is a diagram representing the CNTM's cooperating.

In this example the CNTMs have two accepting states after being state 1. If an algorithm does not receive state and data information from another, even if it has communicated itself, it can transition to state 3 and halt having only ever printed letters from its own alphabet. However if a CNTM receives information from another then its alphabet will be augmented. In this way CNTMs by participating in the simple distributed algorithm of being able to broadcast state and data information to the other CNTMs, are able to change each others behaviour. But because it is not possible to know how a CNTM will read the set M and in which order it will choose its next accepting state, it is not possible to predict the outcome of the computation. This means it is not possible to find an NTM that will perform the same computation. Hence CA's compute a class of effective procedure that an NTM cannot.

Theorem 4.2. *It is proposed that cooperating algorithms perform computations that cannot be computed by a normal NTM.*

Proof. Refer to the formal definitions of a CNTM's and non-deterministic machines above. A non-deterministic machine M is a sextuple $M = (Q, \Sigma, \iota, \beta, \delta, A)$ an CNTM as octuple $A = (Q_k, \Sigma_l, \xi_m, \iota, \beta, \delta, A_n)$.

The states of a CNTM A are given by the set Q_i where $Q_i = Q_1 \cup Q_2 \cup ... \cup Q_p \subseteq Q$ is the unbounded finite set of finite states. Similarly Σ_j where $\Sigma_j = \Sigma_1 \cup \Sigma_2 \cup ... \cup \Sigma_q \subseteq \Sigma$ where $i \leq p$, $j \leq q$ and $i, j, p, q \in \mathbb{N}$is the unbounded finite set of symbols used by

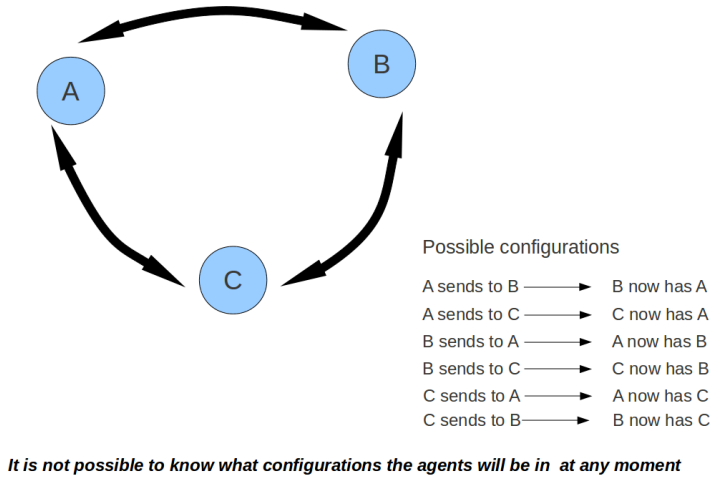


Figure 4.1: Figure 1: Three CNTM's cooperating

the agent.

Let Q_i be the initial configuration of a cooperating algorithm A then as the CA cooperates by the choice transition relation δ new states $\cup Q_a$, Q_b where $a, b \in \mathbb{N}$ will be added so $Q_i = Q_1 \cup Q_2 \cup ... \cup Q_i \cup Q_a \cup Q_b \ldots$

By the same token, by the same application of δ new characters Σ_x , Σ_y where $x, y \in \mathbb{N}$ will be added to the agents alphabet $\Sigma_j = \Sigma_1 \cup \Sigma_2 \cup ... \cup \Sigma_j \cup \Sigma_x \cup \Sigma_y \ldots$

Now an NTM is described as computing a function f from $\mathbb{N}^n$ to $\mathbb{N}^m$ where $n, m \in \mathbb{N}$. Since an NTM is non-deterministic there is only a fixed pool of states and symbols it can compute. With a CNTM on the other hand, it is not possible to know what function an agent is computing as the states and symbols it will compute are not known a priori. Hence an CNTM is not a NTM because while it might compute a partial function f from $\mathbb{N}^n$ to $\mathbb{N}^m$, it is not possible to know which one until it computes it.

□

Theorem 4.3. *It is proposed that proper cooperation can only take place when there*

are three or more cooperating algorithms.

Proof. If there are only two CNTMs, δ the choice relation can only choose elements from its states and symbols or the other CNTM's states and symbols, and so there is no unpredictability. If the states and alphabet of cooperating algorithm A are: Q_i where $Q_i = Q_1 \cup Q_2 \cup ... \cup Q_p \subseteq Q$ and Σ_j where $\Sigma_j = \Sigma_1 \cup \Sigma_2 \cup ... \cup \Sigma_q \subseteq \Sigma$ where $i \leq p$, $j \leq q$ and $i, j, p, q \in \mathbb{N}$

If the states and alphabet of CNTM B are: Q_g where $Q_i = Q_1 \cup Q_2 \cup ... \cup Q_r \subseteq Q$ and Σ_h where $\Sigma_j = \Sigma_1 \cup \Sigma_2 \cup ... \cup \Sigma_s \subseteq \Sigma$ where $g \leq r$, $h \leq r$ and $g, h, r, s \in \mathbb{N}$

Thus, after full cooperation both CNTM's can at most have the following states and alphabets $Q_a \cup Q_i = Q_1 \cup Q_2 \cup \ldots \cup Q_a \cup Q_{a+1} \cup \ldots \cup Q_k$ and $\Sigma_b \cup \Sigma_j = \Sigma_1 \cup \Sigma_2 \cup \ldots \cup \Sigma_b \cup \Sigma_{b+1} \cup \ldots \cup \Sigma_l$ where $a, b \in \mathbb{N}$. Hence the cooperation between two CNTMs is deterministic while three CNTMs cooperating are not. □

4.3.0.3 A proposed definition of an cooperation algorithm community

It is now possible to propose a definition of a community of cooperating algorithms.

$$C_{k,n} = A_1 \cup A_2 \cup ... \cup A_n$$

Where $A_l \in C_{k,n}$ is a CNTM $l \in n$ in community $C_{k,n}$.

There are a number of properties of CNTM with respect to communities. These are the abilities: to join and leave a community; move from one community to another; for a cooperating algorithm to instantiate another CNTM; for a CNTM to terminate another CNTM.

Theorem 4.4. ***Joining a community:*** *Let $C_{k,n}$ be a community and let $C_{l,m} = A_1 \cup A_2 \cup ... \cup A_m$ be another community then $C_{(k+l),(n+m)} = C_{k,n} \cup C_{l,m} = A_1 \cup A_2 \cup ... \cup A_m \cup A_{m+1} \cup ... \cup A_n$.*

Theorem 4.5. ***Leaving a community:*** *Let $C_{k,n}$ be a community and let $C_{l,m} = A_1 \cup A_2 \cup A_m$ be a community such that $C_{l,m} \subset C_{k,n}$ and this subset wishes to leave the community $C_{k,n}$ then there will be two separate communities $C_{k,n} \setminus C_{l,m}$ and $C_{l,m}$.*

Proof. $(C_{k,n} \setminus C_{l,m}) \cup C_{l,m} = C_{k,n}$. □

What is interesting about this is that algorithms can cooperate to produce new and interesting computations that cannot be produced by standard stand alone NTM's.

4.3.1 Extensions, refinements and squeezing

Gödel described Turing's approach of providing a formal mathematical definition for the informal notion of an effective procedure or algorithm, as axiomatic and therefore the correct approach (Shagrir, 2006)(section 3). Turing's method was to model simple calculations called effective procedures (algorithms) that can be performed by an idealised human such as Turing's machines. Indeed in Turing's day, although mechanical computing devices had long been in existence, *a computor* was often taken to mean a person employed to perform arithmetic calculations. It is important to make this distinction because Turing's approach was a logical mathematical approach. He was not trying to describe a physical machine, but was trying to find a way to formally capture our intuitive understanding of algorithms or effective procedures to solve the decision problem posed by Hilbert in 1928 (Hilbert and Ackermann, 1953).

It is interesting to note how this approach proved so effective that, along with the work of many others, it led to the development of the electronic programmable computer. More interesting still is how Turing machines have come to be taken as the exemplars of an idealised electronic computer. It is now studied by all students of computer science and forms the basis of the theory of computation or computability (Goldin and Wegner, 2005). For a fuller discussion also see (Hopcroft et al., 1979) for a classical text on the theory of computation.

Now many theorists think that computability is not adequate to explain modern computers and the internet. To this end, they especially criticise the Church-Turing thesis and have produced many Turing-like machines as well as other formations which they claim somehow breach the "Church-Turing barrier" (Syropoulos, 2008). A common approach is to extend Turing machines with some new property of computers. It is then concluded that there is some mathematical function that a Turing machine

cannot compute by that the new putative machine can. In this way, a counterexample to the Church-Turing thesis is provided claiming it to be false. Goldin and Wegner (2005) argue that modern computers exhibit physical properties such as interaction or unbounded non-determinism that Turing machines do not encompass.

So when Turing found a function that a Turing machine could not compute he was using a mathematical process to discover the limits of algorithms not the power of computers. In 1939 (Turing, 1939) he went on to define *Oracle machines* which have more expressive power than Turing machines. He showed however, that each time a new Oracle is defined it will then be subject to a new version of the decision problem and therefore it will still not able to compute some mathematical problems and that this process could go on ad infinitum.

However it is possible to refine the theory of Turing computability by studying the notion of effective procedures or algorithms. One thing that was missed in the original formulation is that effective procedures can cooperate. In section 4.3 it was shown that cooperation uses the relation, δ, defined above, to enable cooperation. Furthermore that this relation is a kind of choice relation that leads to a computable function that cannot be predicted until is is computed. Not only that, it is possible to generate communities of computations whose outcomes are not predictable. This is novel and interesting and warrants further research.

Finally this work only refines the definition of Turing computability it does not offer a counter example to the Church-Turing thesis. This is because the refinement developed shows that there is a way to improve the definition of effective procedure to account for the possibility of cooperation. To this end the definition of Turing machines was refined to account for this improved definition. According to the thesis there will be many formal machines that will be able to account for this refinement of effective procedures. This will happen because, if the definition of cooperating effective procedure is true,then the resultant Turing machine will also be true. It is important to note there can be any number of refinements to effective procedures which result in suitable formal Turing machine definitions. In this way there is not

a Turing barrier as Goldin and Wegner (2005) claim, just a possibility to refine the Church-Turing thesis ad infinitum.

4.4 Conclusion

The purpose of this chapter is to prove that cooperation can offer computational benefits over stand alone systems. But to achieve this it has been necessary to discuss the contentious foundations of computation. Furthermore agent-based systems are an integral part of this debate. There are many who research computation who argue that modern day computers, including agent-based systems, are not properly explained by computability theory and that a new theory is needed. Goldin and Wegner (2005) argue that interaction between computers is one such property of computers not explained by computability and they offer an *extension* to Turing machines to account for this. There are those such as Cotogno (2003) who argue that those who want a new theory have not understood the power of computability. Furthermore any arguments from the physical reality of computation to computability are flawed as these arguments conflate physical properties with logical ones. Others such as Milner (1982) and Agha and Hewitt (1987) seek to explain these interesting physical phenomena with new theories entirely which do not try to prop up computability. In this context *unbounded non-determinism* has been proposed as a physical property of communication that occurs in agent-based and interacting computers.

It is proposed in this chapter that agent-based systems undoubtedly exhibit unbounded non-determinism. It is agreed that this is a physical property that cannot be explained by computability theory. This is enough to fulfil the goal of this chapter to show how agent-based systems differ from standalone systems. However, it is also possible to offer a new logical-mathematical explanation that, while not offering an extension to the computability, does refine the theory with respect to cooperating algorithms. To this end a new theory of cooperating Turing machines is developed. As a consequence of this theory, it is proposed that cooperating algorithms do add a new dimension to Turing machines, but it is argued any extension or refinement to

Turing machines will not change the Church-Turing theory or the core augments of computability.

CHAPTER V
THE AGENT-BASED FRAMEWORK FOR COOPERATIVE SEARCH IN COMBINATORIAL OPTIMISATION

5.1 Introduction

In this chapter the proposed framework is discussed in detail. 5.2 explains how the framework operates to solve a problem. It also discussed how the agents work. While section 5.3 examines the meta-heuristics and local search heuristics that are available for the the agents to use on the platform. The pattern matching protocol which enables the agents to cooperate is discussed in section 5.4. This involves discussing the proposed model of combinatorial optimisation developed for the framework. The section 5.4.1.1 shows how the proposed ontology implements the model and raises the level of generality of the framework. The chapter is ended with a few concluding remarks in section 5.5.

5.2 Framework architecture and operation

Cooperative search provides a class of strategies to design more effective search methodologies through combining (meta-) heuristics for solving combinatorial optimisation problems. This area has been little explored in operational research. In this thesis, a general agent-based distributed framework where each agent implements a (meta-) heuristic is proposed. An agent continuously adapts itself during the search process using a cooperation protocol based on reinforcement learning and pattern matching. Good patterns which make up improving solutions are identified and shared by the agents. This agent-based system aims to raise the level of generality by providing a simple flexible framework to deal with a variety of different problem domains.The proposed framework has been so far tested on permutation flow-shop scheduling, travelling salesman problem and nurse rostering instances yielding promising results.

The framework makes use of two types of agent: *launcher* and *(meta-)heuristic*

agents (Figure 5.1).

- The launcher agent is responsible for queueing the problem instances to be solved for a given domain, configuring the (meta-)heuristic agents, successively passing a given problem instance to the (meta-)heuristic agents and gathering the solutions from the (meta-)heuristic agents.

- A (meta-)heuristic agent executes one of the (meta-)heuristics or combinations of local search heuristics that are available in the system for the problem in hand. The meta-heuristic/ local search combinations and their parameter settings are all defined in a configuration file the agent reads on launching. In this way the agents are able to conduct searches using different meta-heuristic and local search combinations with different parameter settings.

The framework conducts a search where each agent is launched and registers with the the JADE platform that hosts the framework. Once this is complete the agents are in a wait state waiting for the launcher agent to read in problem from file and to run its seed heuristic and then send the problem to each of the meta-heuristic agents. Only when the meta-heuristic agents receive that problem from the launcher do they embark on a search. Figure 5.1 illustrates this process.

A search has at its core sequences of messages passed between the meta-heuristic agents. Each message is sent as a consequence of internal processing conducted by an agent. A succession of processing and message passing conducted by the agents according to a fixed pattern, amounts to what is, in effect, a distributed algorithm.

In terms of how this was developed, this distributed algorithm can be broken down into message passing implemented as a sequence of interaction protocols which are themselves collection of complex behaviours (See section 3.4.1). The internal processing is conducted as part of an interaction protocol sequence. Typically as part of a protocol an agent receives a message. As a consequence it will perform some internal processing and as a result the protocol will send a response message. The

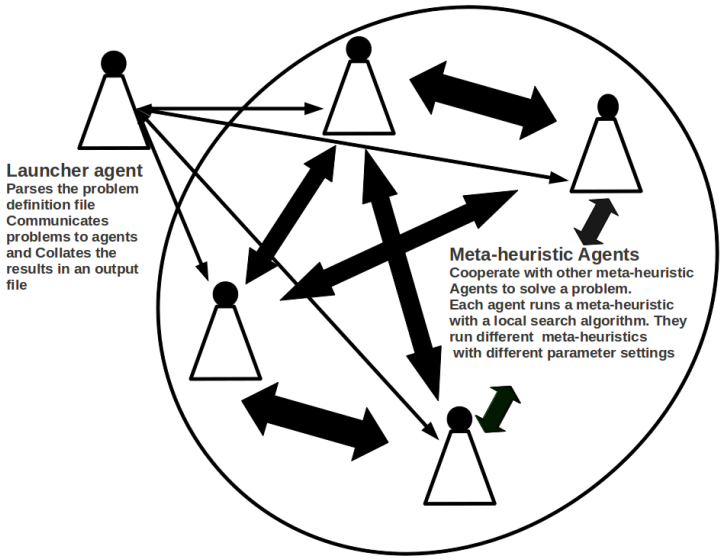


Figure 5.1: The generic multi-agent framework

JADE protocols and behaviours used by the framework can be found in Bellifemine et al. (2007).

One completed sequence of message passing and processing is called a *conversation* and is based upon the contract net protocol (Specification, 2003). The contract net protocol used by JADE complies to the FIPA standard for this protocol (Specification, 2003). It is a very famous interaction protocol (see chapter III) where an initiator agent asks other agents participating in the protocol to offer proposal for solving an initiators problem. The initiator may accept a proposal and reject the rest or it may reject them all. If a proposal is accepted the participator agent performs the proposed action and informs the initiator it has been done and sends a result if there is one. Fig 5.2 is an illustration of the contract net protocol. Here n is the number of agents including the initiator, while m is the number of participants but not the initiator. i is the number of refusals to participate in the protocol and j is the number of participants offering proposals to the initiator. Finally k is the number of of rejected proposals and l (usually 1) are the accepted proposals. How the contract net protocol is used by the framework is described in detail in section 5.4.3.

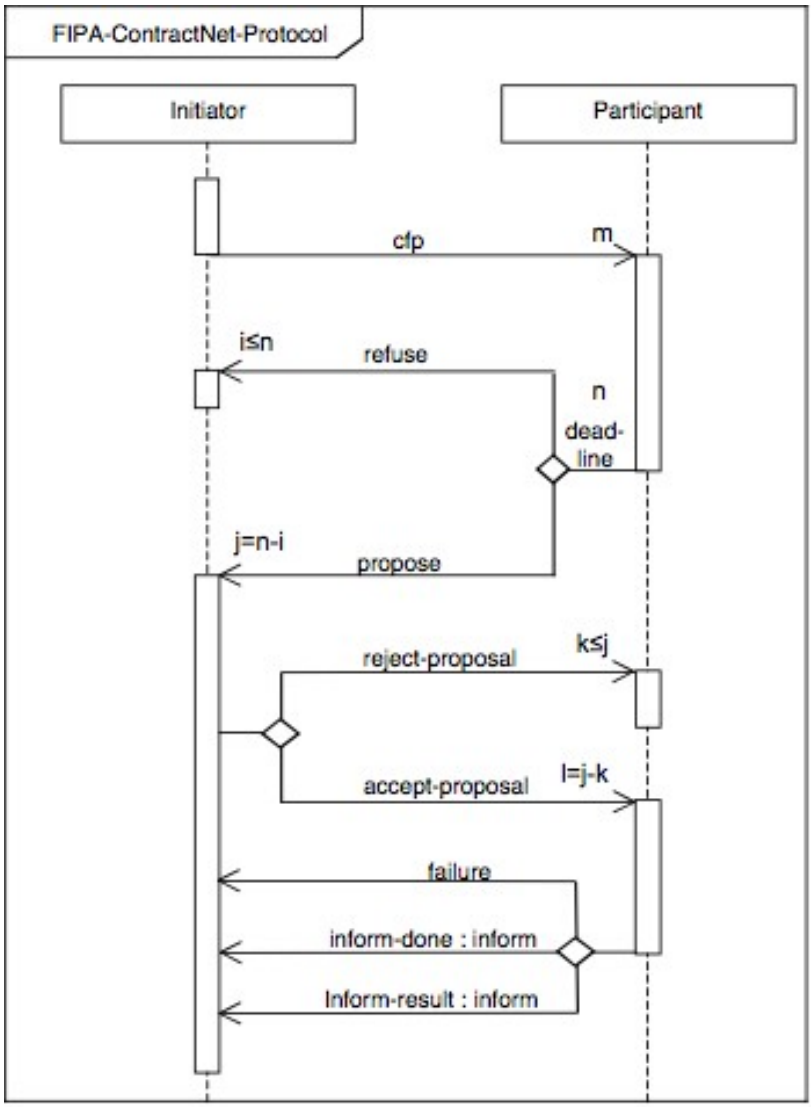


Figure 5.2: FIPA Contract Net protocol

In the course of a search an agent will conduct many such conversations. The (meta-)heuristic agents do not include the launcher agent in their conversations. Only when they have finished their conversations do they then send their best solutions back to the launcher. The launcher chooses the best answer and writes it into an output file. The launcher then moves on to the next problem.

As already explained, there are only two types of agent defined in the framework. The launcher has two specific task of reading problems starting the meta-heuristics

agents off and a search and collecting and presenting results task, at the end of a search. The meta-heuristic agents conduct searches by having conversations with each other. However, looking at Figure 5.3, irrespective of whether it is a launcher or (meta-)heuristic agent, it can be seen that an agent's role during a search is defined by reading in a configuration file when it launches.

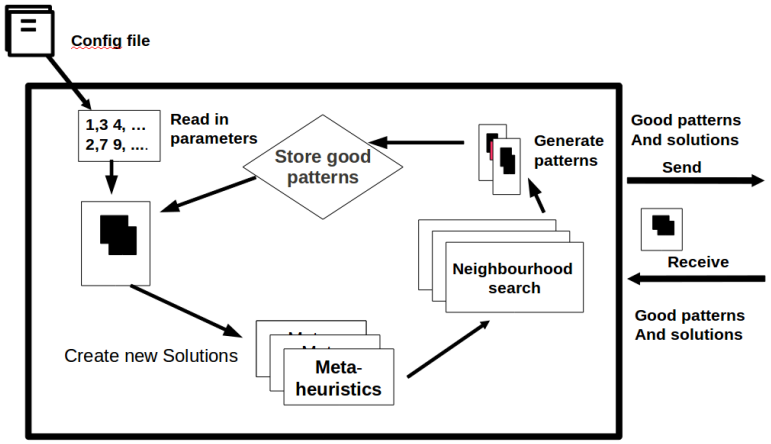


Figure 5.3: A (meta-)heuristic agent from the framework

The configuration file of a launcher agent lists which problems are to be solved. It also contains how many conversations the meta-heuristic agents are going to conduct for a particular problem. A local configuration file is used by each (meta-)heuristic agent at start-up to determine the meta-heuristic/local search combination and parameter settings it will employ. The (meta-) heuristic agents then perform the search based on the problem information sent by the launcher.

Logging facilities can be specified in the configuration files. The system is able to print out the progress of the search on the screen or write it in a log file. One interesting facility is the ability for each (meta-)heuristic agent to log the progress of the search and then have it written to the main log file.

A launcher agent configuration file has the following information:

- Name of the file of the problem to solve.
- Number of (meta-)heuristic agents to be used.
- Number of conversations to be conducted.

A typical (meta-)heuristic agent configuration file has the following information:

- the meta-heuristic to be instantiated by the agent. The choice is tabu search (Glover, 1990a) or simulated annealing (Bertsimas and Tsitsiklis, 1993a). These are very basic implementations of these algorithms.
- the number of iterations the meta-heuristic will perform.
- the size of the tabu list.
- the temperature function to be used for simulated annealing: geometric or logarithmic.
- the local search heuristic to be instantiated and run in conjunction with the chosen meta-heuristic. The choices are 2-opt (used mostly with STSP) or a hill climber with the following moves: swap, insert, reverse and shift.
- the debug information and search progress printed to file.

The framework is designed to be generic and very flexible. This means that all the information needed to apply the system to different problem domains is given either in problem data files or in the launcher and (meta-)heuristic agent configuration files.

Once the meta-heuristic agents have completed the required number of conversations they each send their best result to the launcher agent. The launcher then chooses the best one and then prints an output file with the result solution and objective function value. This file can either be a text file or an XML as required by the problem definition.

5.3 (Meta-)heuristic agents

There are a number of heuristics and meta-heuristics implemented on the platform. Any meta-heuristic agent can be configured to instantiate any of them by supplying the correct codes in the agent's configuration file. The Launcher agent also executes a number of seed meta-heuristics depending on the problem that is being solved.

Table V.1: Seed Heuristics used by the launcher agent implemented for the framework

Problems	Heuristic	Reference
STSP	Nearest Neighbour algorithm	(Flood, 1956)
PFSP	NEH	(Nawaz et al., 1983b)
nurse rostering	Greedy	(Bilgin et al., 2012)

Table V.1 lists the different heuristics and references used by the Launcher agent to produce seed solutions for different problem types. These algorithms have all been implemented especially for the platform in JAVA from the pseudo-code provided by the referenced papers.

The use of seed heuristics on the launcher is beneficial but not necessary. Testing indicates (see table V.2) that the seed heuristics reduce the number of conversations executed by the meta-heuristic agents in order to find good solutions. One reason for this is that the seed heuristic enables the agents to start from a good position in the solution space. The distributed pattern matching heuristic executed by the agents robust. It causes the agents to diversify through the solution space, but starting from a good position through the use of a seed solution increases their chances of finding further good solutions. However if the agents start from random seeds they are already in diverse parts of the solution space and it take longer for them close in on good solutions.

Compare the TSP instances table in V.2 with those in table VI.1. Both tables were produced by taking the average of 10 runs of instances were taken from the well known benchmark library TSPLIB (Reinelt, 1991). The table results are each calculated

taking the percentage increase from the known optimum value. The instances in table V.2 were taken from tests where the meta-heuristic agents all started their search from different random seeds. They also executed 5002 conversations. While the results in VI.1 were obtained over 3002 conversations with tests using a seed algorithm. To this end the results are closer to the optimum and therefore fewer conversations were needed to achieve good results. All other settings for both sets of tests were exactly the same. These results are suggestive that using a seed algorithm in the launcher agent tend to help the meta-heuristic agents achieve better solutions with fewer conversations.

Table V.2: % deviation from optimum of TSP instances with random seed over 5002 conversations

	12 Agents	8 Agents	4 Agents	1 Agent SA	1 Agent Tabu
eil101	3.69	2.70	4.13	10.016	10.016
a280	13.57	14.17	13.06	15.82	15.82
gil262	14.83	14.85	15.29	15.43	15.43
pr299	15.80	16.29	17.33	18.06	18.06
rat575	18.18	18.35	18.65	18.81	18.81

The heuristics and meta-heuristics available to the meta-heuristic agents are all simple classical implementations of famous heuristics which again have been coded in JAVA from the pseudo-code in the referenced papers.

It should be noted that all the heuristics and meta-heuristics implemented on the platform have been implemented according to the framework model and ontology as explained in section 5.4. This means that the (meta-)heuristics instantiated by the agents never work on problem specific data, but on the generic internal representation of combinatorial optimisation problems developed for the framework.

Here is the list of Local search heuristics instantiated by the meta-heuristic agents:

- 2-opt (Beullens et al., 2003)

- Hill-climber with swap
- Hill-climber with random shuffle of sub-list

The hill-climber is a simple perturbation heuristic. It takes a seed solution and seeks to improve it by taking two random numbers modulo the size of the list. It then, depending on the choice made in the agent's configuration file, swaps these two from the seed solution using the two random numbers as the start and end of the list. In each case a population of potential solutions is generated and the objective function of each is evaluated. The best improving putative solution is chosen as the new solution to be passed back to the relevant meta-heuristic.

The 2-opt used is a version of the famous 2-opt first proposed by Croes (1958). However this version (Beullens et al., 2003) also uses neighbour lists and active marking which while it does not improve the overall solution quality, does greatly increase the speed of the search.

Here is the list of Local search Meta-heuristics instantiated by the meta-heuristic agents:

- Tabu Search (Glover, 1990a)
- Simulated Annealing (Bertsimas and Tsitsiklis, 1993a)
- Variable Neighbourhood Search (Currently only used for nurse rostering) Bilgin et al. (2012)

The tabu search agents implement a basic tabu search. In a basic tabu search, the search starts from a seed solution and iteratively moves from the current solution to its best neighbouring solution using neighbourhoods even if that neighbourhood worsens the objective function value. To avoid returning to recently visited solutions, successful moves are declared tabu for a certain number of iterations. The length of the tabu list is specified in the instantiating agent's local configuration file. In addition, an aspiration criterion is defined to accept tabu neighbourhoods with an objective function value better than the best available so far.

The basic simulated annealing meta-heuristic used by the the agents is implemented with a choice of a geometric or logarithmic cooling schedule. The choice is again specified in an instantiating agent's configuration file. The search starts from a feasible solution and iteratively moves from the current solution to its best neighbouring solution. Improved solutions are accepted, while non improved solutions are accepted with a probability $\exp \frac{\Delta}{t}$ where Δ is the change in the objective function value, and t is the temperature which controls the acceptance probability. All the case studies in this thesis use a geometric cooling schedule. It was chosen because it could be made to diversify easily with Δ being set at 0.9 and also it performed more quickly than the logarithmic cooling schedule. The parameter settings for the platform are further explained in section 6.4.1.

The Variable Neighbourhood Search (VNS) and the local search heuristics are part of a nurse rostering model developed by Bilgin et al. (2012) and were only used in conjuction with the nurse rostering case study in chapter VIII. This was taken and re-coded and added to the framework to prove that it is flexible enough to work on constrained problems as well. This means that currently VNS is only available for nurse rostering problems.

The VNS algorithm utilises several local search moves and holds the parameters of the executed moves in a tabu list. The neighbourhoods utilised are:

- **shift**: This assigns a nurse to a new shift to the roster.
- **Delete shift**: This move deletes a shift from the roster
- **Single shift day**: An assignment is removed from a nurses schedule and added to another nurse on the same day if the second nurse has no assignment on that day and has the associated skill type.
- **Change assignment based on compatible shift type**: The shift type of an assignment is changed to another compatible shift type defined in the coverage constraints for the associated day and skill type.

- **Change assignment based on skill type**: This neighbourhood can be used when the nurses have at least two different skill types. It deletes an assignment and adds another assignment to one of the nurses other skill types.

- **General Assignment Change**: An assignment is changed to another shift type where the skill type of the assignment remains the same.

At each iteration of the algorithm, a single move from the list above, is applied. The moves are chosen in random order. Each possible move is validated against the hard constraints (they must be complied with) of a given nurse rostering problem. In this way any potential roster remains feasible throughout the execution of the algorithm. The algorithm also features a tabu list is used to avoid cycling and getting stuck in local optima. The best move allowed by the tabu list is executed. The length of the tabu list is variable during the execution. It is increased at each non-improving iteration and decreased if there is an improvement.

5.4 Cooperation by pattern matching and reinforcement learning

In this section the pattern matching protocol used by the meta-heuristic agents is described. In order to do this it is necessary to explain the proposed model for combinatorial optimisation used throughout the framework.

In chapter IV it has been shown that cooperative agent computation has theoretical benefits. With this in mind, a generic model for multi-agent cooperative search in combinatorial optimisation is proposed.

5.4.1 Combinatorial optimisation ontology

Many combinatorial optimisation problems can be modelled as hierarchy of problems with the general non-linear programming problem at one end and linear programs and flow and matching problems at the other (Papadimitriou and Steiglitz, 1998). The following equations describe a general mathematical model for combinatorial optimisation problems.

$$\text{maximise or minimize } (\mathbf{c}^T\mathbf{x}) \tag{5.1}$$

$$\text{subject to } \mathbf{Ax} \leq \mathbf{b} \tag{5.2}$$

$$\text{and } \mathbf{x} \geq 0 \; where \; \mathbf{c}, \mathbf{x}, \mathbf{b} \in \mathbb{R}^m, \mathbf{A} \in \mathbb{R}^{m \times n} \tag{5.3}$$

Here $\mathbf{c}$ is a cost vector of coefficients and $\mathbf{x}$ is a vector of decision variables. $\mathbf{A}$ is a matrix of constraint coefficients and $\mathbf{b}$ is a vector of coefficients. Equation 5.1 describes a cost vector and a decision variable vector where the overall cost of multiplying these two vectors together is to be optimised. Equation 5.2 defines a matrix of constraints, while equation 5.3 represents the legal values for the decision variables.

If equation 5.1 is a non-linear function then the whole problem is called non-linear. Obviously if equations 5.1 and 5.2 are linear then the problem is called linear, but if equation 5.3 requires the decision variables to be integers then the problem is known as an integer problem. The problems examined in this thesis are modelled as integer problems. To this end equation 5.3 is re-written as follows:

$$\text{and } \mathbf{x} \geq 0 \; where \; \mathbf{c}, \mathbf{b} \in \mathbb{R}^m, \mathbf{A} \in \mathbb{R}^{m \times n} \; and \; \mathbf{x} \in \mathbb{N}^n \tag{5.4}$$

The constraints defined in equation 5.2 check the validity of the decision variables with respect to the problem being optimised. In the case nurse rostering case study of chapter VIII, it has many complex constraints representing the many complex employment and contractual issues involved in assigning nurses to particular shifts. Some examples of these include the constraint that a nurse may only be assigned to one shift a day and a nurse cannot be assigned to a day shift the day after night shift. Mathematically these constraints would be modelled as a long list of inequalities and characteristic functions.

$$x_i \in \mathbf{x} = \begin{cases} 1 \text{ if } x_i \text{ is assigned to a solution} \\ 0 \text{ if } x_i \text{ is not assigned} \end{cases} \quad \text{and } i \in \mathbb{N} \tag{5.5}$$

By studying this mathematical model a general method for modelling its algorithmic structure by using ontologies can be determined.

5.4.1.1 Ontologies

Ontologies (see section 3.4.3.1) play an important role within the framework. They define a set of general representational primitives to model combinatorial optimisation problems and as such are semantic (Gruber, 1993). A number of representational primitives are defined with respect to the mathematical model in section 5.4. These are used by the agents to both share information with each other and by the heuristics employed by the agents to solve a problem. This makes the agents very flexible because many combinatorial optimisation problems can be modelled with these primitives and so the agents can solve problems without recourse to problem specific details. To substantiate this claim the implemented system was tested on travelling salesman, permutation flowshop scheduling and nurse rostering problems. These results are discussed in chapters VI, VII, and VIII respectively.

A possible solution to a combinatorial optimisation problem is often represented algorithmically as permutation of decision variables (equation 5.3) which is optimised according to an objective function (equation 5.1). From this permutation the costs of assigning certain decision variables to a possible solution can be ascertained by looking them up in a table of costs. Finally the possible solution is verified, according to any constraints associated with the problem which have been represented algorithmically as a collection of verification rules with which any potential solution has to comply (equation 5.2).

The ontology used by the framework generalises these notions as abstract objects. Any potential solution is called a *SolutionData* object while any constraints are modelled by the *Constraints* abstract object. The elements of any possible solution are

called *SolutionElements* objects. Finally, another abstract object used by the ontology is called a *HeuristicData* object. These are just pairs of SolutionElements objects. Referring to the mathematical model above, HeuristicData objects represent pairs of decision variables. These are used by the pattern matching protocol employed by the framework. This is explained in section 5.4.2 below.

The ontology is general enough to handle many problems in combinatorial optimisation, but are currently used to solve problems in STSP, PFSP and NR. To this end,currently, the primitives of the ontology are used to represent: a tour, a job schedule, or a nurse roster as a *SolutionData* object; a pair of cities, a pair of jobs on two machines, or a pair of nurse assignments as *HeuristicData* objects; any constraints are represented as *Constraints* Objects; and finally, individual cities, jobs or nurses are represented as *SolutionElements* objects.

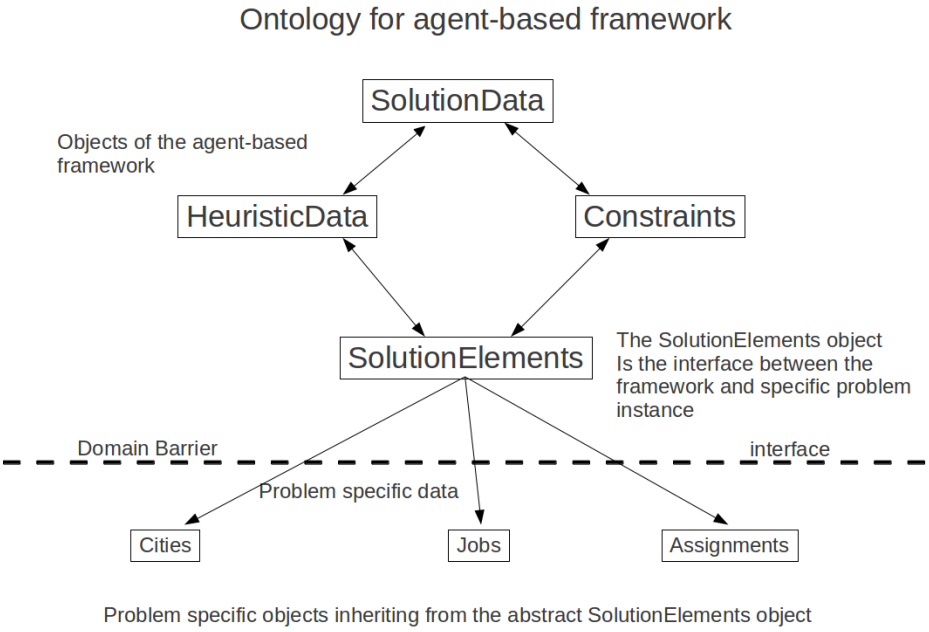


Figure 5.4: The combinatorial optimisation ontology

The ontology of figure 5.4 consists of four elements.

- **SolutionElelments:** A SolutionElement is an abstract object that can repre-

sent a problem specific object such as a *job* in PFSP or a *city* in STSP or the *assignment* of a nurse with specific skills to a shift on a certain day as required by a given nurse rostering problem.

- **HeuristicData:** A HeuristicData object contains two SolutionElements objects. These are currently used to represent pairs of *cities, jobs* or *assignments* in a permutation that will be in the cooperation protocol to identify good patterns in improving permutations.

- **Constraints:** The Constraints interface is the interface between the high level framework and the concrete constraints used by a specific problem. These are used to verify a valid permutation.

- **SolutionData:** A SolutionData object is a list of SolutionElements objects and therefore is the permutation that is optimised by the framework. It currently represents a *schedule of jobs* PFSP or a *tour* in STSP or a *roster in NR*

All message passing on the framework including the whole ontology is represented in XML. This can be very advantageous as many benchmark problems, these days, are also represented in XML. This makes the interface between problem definition and ontology seamless in practice. Fig 5.4 shows the structure of the ontology and how SolutionElements are the interface between the framework and a concrete problem.

The SolutionElements, HeuristicData, Constraints and SolutionData objects are used also by the agents to facilitate inter-agent cooperation. The pattern matching protocol, the meta-heuristics and local search heuristics used by the agents in the framework all manipulate these objects only. This is similar to the so called "domain barrier" in hyper-heuristics (Ouelhadj and Petrovic, 2010). However, here the interface between the generic framework objects and the domain specific objects, is set at the decision variables level in this framework, unlike hyper-heuristics which sets its "domain barrier" at the level of heuristics.

5.4.2 Cooperation protocol with pattern matching and reinforcement learning

The framework makes use of a pattern matching protocol which is used by the agents to enable cooperation between them. It is, in effect, a distributed meta-heuristic which uses the ontology primitives rather than any problem specific data. One iteration of this distributed meta-heuristic will be called from now on a *conversation*. The implementation specifics of this process are discussed in section 5.4.3. In the course of a search the agents may carry many conversations. References to parts figure 5.5 are provided to show how this more theoretical description of a conversation relates to its practical implementation.

The search proceeds with intensification and diversification phases. At the start, the launcher provides each agent with a permutation of SolutionElements objects whose ID's are represented as integers. For example, take the permutation where $n = 10$: with the following ID's: $2, 4, 7, 6, 5, 8, 9, 0, 1, 3$. The agents then try to intensify it using their given meta-heuristic/heuristic combinations (see I_1 and R_1 in figure 5.5).

Next, in the diversification phase (see I_2 and R_2 in figure 5.5), each agent splits their best-so-far permutations into pairs of HeuristicData objects. Given the permutation above, it is possible to generate n pairs from it:

$$(2,4),(4,7),(7,6),(6,5),(5,8),(8,9),(9,0),(0,1),(1,3),(3,2)$$

.

Again the integers represent the ID's of each SolutionElement object. One pair represents a HeuristicData object.

This is done as a way of breaking down a permutation into patterns that are of the same length while retaining the basic order of the permutation. This is the simplest method for doing this; other pattern lengths such as triplets and quadruplets or mixtures will be discussed in section 5.4.2.1 below.

The agents can then each compare pairs generated from their own best-so-far

permutation with pairs generated by the other agents. In any conversation the agents take on one of two roles: initiator and responder. These roles are not predetermined but are assigned as the conversation unfolds. This will be explained in section 5.4.3.

After each agent has generated its list of pairs, the initiator collects all the pairs (see I_3 in figure 5.5) from the other agents and scores them based on how frequently they appear. Only those pairs which occur as frequently as the number of agents in the conversation, have their relative costs calculated. Table V.3 shows the example of 4 agents cooperating. Here the pattern (2,3), highlighted in bold, occurs with the same frequency as the number of agents, in this example, four times. For this reasons this pattern is retained by the initiator agent. Remember, each integer in a pair is also the ID of a SolutionElements object in the problem at hand. The ID is used to look into the cost matrix to calculate the cost of the pair with respect to the objective function. In the case of STSP this is distance from one city to another, while for PFSP it is the CMAX (Nawaz et al., 1983b) of pairing one job with another across a given number of machines and for NR it the cost of a particular pair of nurse assignments.

Table V.3: The initator collects all the pairs from the agents and only uses those that have occur all in agents

Agents	Patterns
Agent 1	**(2,3)**(5,1)(7,4)
Agent 2	(1,2)**(2,3)**(4,5)
Agent 3	**(2,3)**(7,4)(9,7)
Agent 4	(6,7)(5,8)**(2,3)**

Once the best pairs have been identified and costed, they are shared out amongst the agents (see R_3 in figure 5.5). Each agent also maintains a pool of good pairs found so far. When an agent receives a new consignment of good pairs it compares these with the ones it already has in its pool.

Once this process is complete each agent tries to build a new permutation using the pairs in its pool (see I_4 and R_4 in figure 5.5). It does this by trying to link pairs which share the same second digit and first digit. For example, if the following pairs are found in the agent's pool: $(2,5)(3,6)(5,3)(1,7)$. The agent then tries to link them as follows: $(2,5)(5,3)(3,6)$ leaving $(1,7)$ unlinked. It then builds the new permutation by taking the linked pairs and generating the first part of the permutation by removing any repeats of numbers as follows: $(2,5,3,6)$. Any unlinked pairs not used so far fill in the next part of the permutation giving: $(2,5,3,6,1,7)$. Finally, any numbers missing from the permutation are taken from the agent's best-so-far permutation. This is done because a permutation must be a list of *all* the ID's in use in a problem. The new permutation produced is the result of the diversification phase and will be used by the agent as its new best-so-far permutation. It also marks the end of a conversation and the new permutation is ready for the start of a new conversation.

5.4.2.1 Discussion of cooperation strategy

As already mentioned, this simple distributed pattern matching meta-heuristics, like most meta-heuristics, has an intensification and diversification phase. Broadly, intensification occurs when an agent applies its given meta-heuristic, while the pattern matching and greedy heuristic acts as a diversification phase. However it is important to qualify this statement given that most meta-heuristics already have intensification and diversification phases anyway. The difference is that this distributed meta-heuristic is robust: different agents will be working on different parts at or in a given search space, at the same time.

To this end it is necessary to thoroughly search a given area of the search space, using perturbation meta-heuristics, to identify a local neighbourhood finding the best minimum in that area of the search space. This is the intensification with respect to a distributed search. The diversification phase must not only be robust in getting the agents to search different parts of the search space, but it must also provide a method for the agents to converge to good solutions, otherwise the strategy would

involve nothing more than parallel random starts.

The pattern matching part of the heuristic is based on two assumptions: different meta-heuristics have different strengths and weaknesses; and, good solutions will share reoccurring features. The meta-heuristics on the platform use very basic forms of SA(Bertsimas and Tsitsiklis, 1993a) and TS (Glover, 1990a) and VNS (Bilgin et al., 2012). The SA uses a geometric cooling schedule that cools slowly so that the algorithm will diversify more often. The TS, on the other hand is set with a small fixed tabu tenure of seven causing the algorithm to be more intensive. VNS uses either a random or looking-ahead choice functions for determining which local search heuristic to use next.

The local search algorithms used are a hill-climber algorithm using a swap move for PFSP and 2-OPT for TSP. These are chosen as they are very well known in the literature as local search techniques best suited to these two problems. Indeed it is the intention for the generic framework to use problem specific local search heuristics. The reason for doing this is that they have a large bearing on how a search will progress and it is relatively easy for each to write and add a new local search heuristic, given the generic modular framework provided by the platform. However they still optimise using the permutations of ID's described above.

The idea behind setting TS to be intensive and SA to be diverse is to increase the difference between the two algorithms thus increasing coverage of the search space. Also, for the pattern matching algorithm to be effective, new patterns must be generated. Therefore, by adopting this strategy, the chance of finding more patterns is increased. It is still subject to further testing as to how effective this strategy is, but it seems like a good early strategy to adopt.

It is an assumption of this work that good features of improving solutions will reoccur if different meta-heuristics are working in parallel. However, it is important to find a method of identifying good patterns. Given a permutation N there are $2^{|N|}$ possible subsets that could be seen as features of the permutation. For large permutations this is an impossible number of features to search. It was decided

therefore, just to look at the best-so-far permutation produced by an agent's meta-heuristic and try to identify features that preserve its current order and the costs implied between the different members. The simplest way to do this was to break the permutation up into pairs maintaining its order as explained in section 5.4.1. It is intended to explore different pattern lengths and mixed pattern lengths in future work.

The cooperation protocol shares only those patterns that have a frequency that is the same as the number of meta-heuristic agents. These patterns are also costed with respect to the objective function and only those that improve on patterns that share an element are kept. This ensures that each agents pool does not get too big and only those patterns that have an improving distance are kept. Having updated its pool of pairs each agent tries to build a new permutation according to the process described in section 5.4.1. Because of the scoring and pool updating process there are unlikely to be enough pairs to build a new permutation from the pool alone. Therefore the agent's current best-so-far permutation is used to supply the rest of the numbers needed. This is a diversification process: the good patterns mean that the new permutation does not stray too far from good solutions but at the same time it is a wider diversification process than that used by the agent's internal meta-heuristics such as SA or TS. This is because, as mentioned earlier, an agent's own meta-heuristic will search a local area and diversify from local minima so a wider diversification process is needed to get each agent to search a new area of the search space.

The new permutation generated in this way, is similar the seed solutions commonly used by researchers when using a meta-heuristic to tackle a specific problem. The seed is generated typically by a greedy heuristic, so that the meta-heuristic in question starts from a good position in the search space. By the same token, the patterns found by the agents provide the basis of new seed solutions that can then be intensified by an agent's meta-heuristic at the start of the next conversation.

The pattern matching phase employs a simple form of re-enforcement learning.

This is achieved by allowing only those patterns that occur in permutations of all agents during a conversation to be kept. These are stored in a pool of good patterns where new ones are allowed only if there is an improvement on a previous pattern. Thus the system only allows a limited number of improving patterns that can be used from one conversation to another. They form the basis of a new permutation generated by a diversification process. The idea is that these patterns ensure any new permutation is not so significantly worse that it hinders the overall progress of the search.

In future work, tests will be carried out on the frequency of patterns and on the size of the pool. The permutations and pairs discussed in this section are implemented by using ontologies. This will be discussed in the next section.

5.4.3 Conversation

Figure 5.5 shows the pattern matching protocol used by the (meta-)heuristic agents. One complete execution of the algorithm illustrated is a *conversation*. In any conversation, there will be a (meta-)heuristic agent that takes on the role of an initiator and the others are responders. Any (meta-)heuristic agent can be the initiator, but it is determined in the previous conversation which agent will be the initiator for the current conversation.

In figure 5.5 an agent taking on the role of initiator starts a conversation. It takes a new permutation either generated from a previous conversation or a seed supplied by the launcher agent. The new permutation or solution is then improved by the chosen (meta-)heuristic for that agent. In figure 5.5 this is the box numbered I_1. When an improved solution is generated, it is sent to the other (meta-)heuristic agents and this marks the start of the conversation. In I_2 of figure 5.5 the initiator breaks its best-solution-so-far into edges or pairs, as explained above in section 5.4.1.

On receiving the initial solution from the initiator, the other (meta-)heuristic agents also take their best-so-far permutations, R_1 in figure 5.5, using their designated (meta-)heuristics. They too each break their best-so-far solutions into edges or pairs,

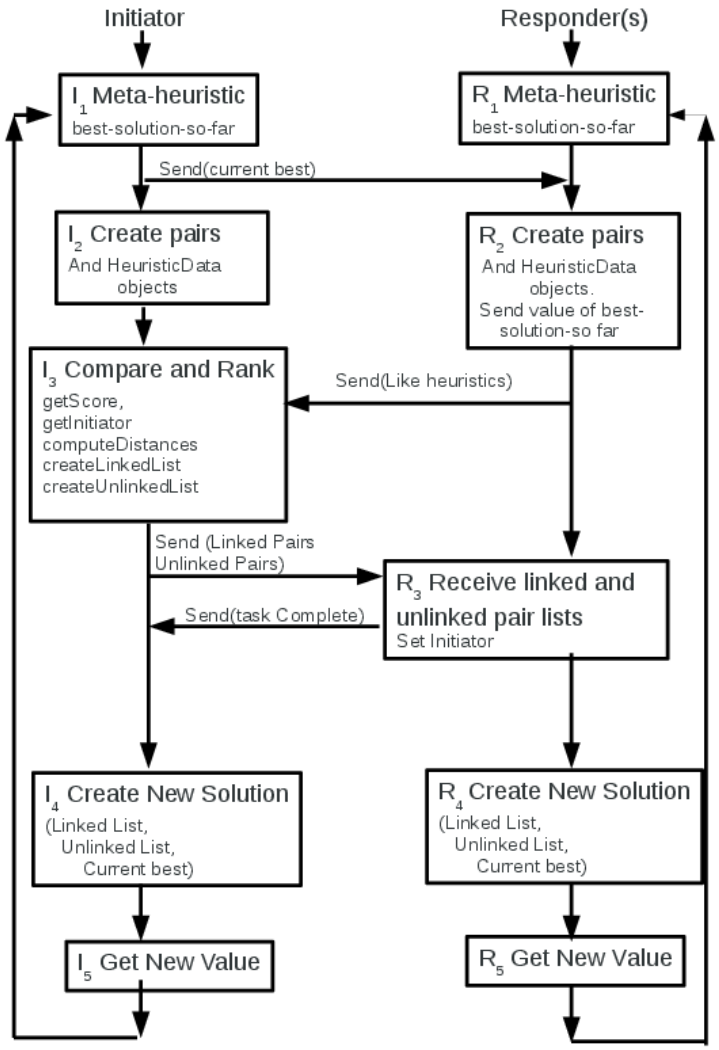


Figure 5.5: The Cooperation Protocol showing one iteration of a conversation

I_2 of figure 5.5. HeuristicData objects are created from these pairs storing the first and second elements of the pair.

The initiator creates its own HeuristicData objects from the pairs generated, I_2 of figure 5.5, while the responder (meta-)heuristic agents send their HeuristicData objects to the initiator. The receiving (meta-)heuristic agents also send the value of their best-so-far solution. This will be used by the initiator to determine which (meta-)heuristic agent will be the new initiator in the next conversation (see R_2).

In I_3, the initiator receives the HeuristicData objects from the responding agents and collects them together. Each HeuristicData object is scored and ranked based on frequency by the initiator. This can be seen in box I_3 of 5.5 as the functions *getScore* and *computeDistances*. Tests show that it is best to accept only HeuristicData objects with a frequency score the same as the number of (meta-)heuristic agents (algorithm 1 getScore).

Each agent keeps a pool of HeuristicData objects which is made up of only of high scoring HeuristicData objects. When new objects are found during course of a conversation then the agent will update according to the following strategy. Each new high scoring pattern is compared with those already in the pool. If a new pattern shares the same first or last number, but the distance between the two elements of the pattern is better than that of the pool, then the new pattern replaces the old pattern. It functions as a short term memory keeping a limited list of good patterns.

Input: List of HeuristicData Objects
Output: Set of HeuristicData Objects
foreach *HeuristicData object in List* **do**
    score = frequency of this HeuristicData object in List;
    if *score >= number of agents* -1 **then**
        Object.setScore(score);
        output.add(object);
    end
end

Algorithm 1: getScore: an algorithm for scoring patterns

In figure I_3 of 5.5 the initiator using the function *createLinkedList* then tries to build a linked list. To do this it uses the high scoring HeuristicData objects found in the current conversation and with the patterns in the pool. For example, if the pool contains the following HeuristicData objects with first and second elements expressed here as pairs (4,7) (6,1) (7,2) (2,6) (5,9) (3,8), the linked list generated from the HeuristicData objects will have the following order (4,7) (7,2) (2,6) (6,1). Algorithm

2 shows the pseudo code for this mechanism. Any HeuristicData objects not linked in this way are stored in an unlinked list (5,9) (3,8).

```
Input: Set of scored HeuristicData Objects
Output: Set of linked HeuristicData Objects
copy the scored list;
while copy is not empty and output is not unchanged do
    foreach HeuristicData object in scored set do
        if output set is empty then
            output.add(object);
            copy.remove(object);
        end
        if the first element of the object equals last element of the last object in
        the output then
            add object to the back of the output set;
            copy.remove(object);
        end
        else if the second element of the object equals the first element of the
        first object in the output then
            add object to the back of the output set;
            copy.remove(object);
        end
    end
end
```

Algorithm 2: createHeuristicLinkedList: an algorithm creating a linked list of HeuristicData objects

In I_3 of figure 5.5, through the function *getInitiator*, the initiator also determines which (meta-)heuristic agent is going to be the initiator in the next conversation. This is done by taking all the values of the best-so-far solutions sent by each (meta-)heuristic agent and then identifying which (meta-)heuristic agent has the best ob-

jective function value. The (meta-)heuristic agent with the best value will be the new initiator in the next conversation. The initiator then sends the linked and unlinked lists of HeuristicData objects to the receiving (meta-)heuristic agents. In the same message it also indicates which (meta-)heuristic agent will be the new initiator in the next conversation.

The other (meta-)heuristic agents receive the lists of HeuristicData objects from the initiator (see box R_3 in figure 5.5). They also update their internal pools as described above. In box I_4 and R_4 of figure 5.5, both initiator and responder (meta-)heuristic agents then each create a new solution by updating their internal pools and both the linked list and the unlinked list sent to them. They also use their current best solution produced by their own meta-heuristic . The new solution is created by trying to build first a list of numbers from the linked HeuristicData objects. The unlinked Heuristic Data objects are used next to supply more numbers. Finally the (meta-)heuristic agent's best-so-far solution provides any missing numbers. In the example I_3 the createLinkedList function created the following linked list (4,7) (7,2) (2,6) (6,1) with the (5,9) and (3,8) as the remaining unlinked objects. Therefore on I_4 and I_4 in figure 5.5 the following solution would be created by one of the agents. The linkedlist and unlinked list would produce the following incomplete list (4,7,2,6,1,5,9,3,8). As our example list is length 10 then the numbers 7 and 10 missing from our incomplete list. Supposing the best-solution-so-far from the agents meta-heuristic was (3,5,2,7,8,4,1,10,9,6), then the numbers 7 and 10 occur in the order (7,10) in this list. This means the missing numbers in our new list will be added as (7,10) at the end of our list. The complete list is therefore (4,7,2,6,1,5,9,3,8,7,10). In this way, for each agents, a new unique permutation is generated and the objective function value is calculated (see boxes I_5 and R_5).

Whichever (meta-)heuristic agent was deemed to be the initiator from the current conversation will be the new initiator for the next conversation. The process repeats and continues until the number of conversations set from the launcher agent is completed.

The framework described here uses ontologies to model problems in combinatorial optimisation and provides a number of representational abstract primitives so that the agents are never concerned with problem specific data. This makes the framework generic and modular. It is possible to add new functionality to the system as long as the framework primitives are integrated with problem specific primitives. Indeed this was done for the implementation of the nurse rostering problem described in chapter VIII where a constraint verification system developed by other researchers was added to the system as a module so that the system could solve nurse rostering problems. The agents still only manipulated the ontology's primitives.

The framework also uses a distributed algorithm with the agent's cooperation protocol. They use this protocol to share good features of different problems to enable more efficient search. Testing has shown that the protocol acts like a diversification strategy in the early stages when the agents share patterns and then they quickly intensify finding better solutions. This will be explained in greater detail in chapters VI, VII and VIII.

5.4.4 A theoretical note

In chapter IV it was proposed that in emergent distributed algorithms agents cooperate by each playing an equal part in the algorithm. This is only possible if there was no overall control process. Also each participant must be able, if not cooperating, to independently solve the problem at hand. In short they are autonomous. The cooperation protocol used by the agents in the framework, just described in this section meets all these requirements and therefore is an example of the type of the emergent distributed algorithm.

5.5 Conclusion

An agent-based framework for cooperative search has been proposed The proposed combinatorial optimisation ontology developed for the framework makes it easy to develop and implement new heuristics and meta-heuristics thus raising the level of

generality. It also means that other (meta-)heuristics developed by other researchers can be easily integrated into the system as will be seen in chapter VIII. The use of JAVA means that the platform can be ported from one operating system to another with no change to the system. Indeed testing of the case studies in chapterVI on STSP, chapter VII on PFSP and chapter VIII on NR, was done on a mixture of networks, some running Windows 7 and some running Linux. The pattern matching protocol is an example of an emergent distributed algorithm. The agents do not share full solutions but knowledge about good features of potential solutions identified by statistical frequency. Finally, this framework will be published as an open source project so that other (meta-)heuristics and cooperation protocols can be added and tested by other researchers. The project is called MACS (Multi-agent Cooperative Search) and will be published at the following link http://www.port.ac.uk/maths.

CHAPTER VI
THE TRAVELLING SALESMAN PROBLEM

6.1 Introduction

This chapter describes the first of the case studies undertaken to test the agent-based system for cooperative search. It was conducted on the symmetrical travelling salesman problem. In section 6.2 a short history of the TSP is provided. This is followed by a mathematical description of the Symmetrical Travelling Salesman Problem (STSP) in section 6.3. The general experimental settings not for just the STSP but all the case studies of this thesis are provided in section 6.4.1. Specific STSP experimental settings are introduced in section 6.5. The STSP benchmarks used for testing are introduced in section 6.6. In section 6.7 the results of the tests are presented and analysed. Finally some conclusions are offered in section 6.8.

6.2 A brief history of the TSP

In chapter II, it was explained that there have been a number of problem specific multi-agent platforms developed, to try and solve specific problems in combinatorial optimisation (Xie and Liu, 2009; Vallada and Ruiz, 2010); and also, there have been more general systems proposed able to tackle many combinatorial optimisation problems (Talukdar et al., 1998; Malek, 2010). However, these latter systems have taken different approaches to the work described here: such as breaking big problems into different jobs and letting the agents solve these (Talukdar et al., 1998) or letting the agents only use one meta-heuristic to solve problems (Malek, 2010). It was established also that as far as it is known there is no framework and platform that is asynchronous, that uses different meta-heuristic agents and ontologies to solve hard combinatorial optimisation problems.

As Laporte (2006) has shown, the TSP has it roots in antiquity, but the first modern formulation of STSP and its solution by exact methods, was provided by Dantzig et al. (1954). Other important historical articles are Land and Doig (1960);

Little et al. (1963) which introduce the Asymmetrical Travelling Salesman Problem (ATSP) using exact solving methods. The first modern use of heuristics for NP-hard problems in operational research can be traced back to the work of Croes (1958); Lin (1965). Also it is important to mention the famous algorithm of Lin and Kernighan (1973b) which continues to be the most effective heuristic algorithm for obtaining impressive results for all sizes of travelling salesman problem .

A recent agent-based approach to solving the TSP is the Multi-Agent Optimization System for solving the TSP (MAOS) by Xie and Liu (2009). The MAOS system uses a population based evolutionary approach to solve the TSP. Each agent has some simple rules that enable it use simple hill-climber heuristics to manipulate potential solutions and share them with the other agents. The results are comparable with those of the Lin-Kernighan algorithm. Other agent-based systems using population based techniques such as ant colonies and genetic algorithms are Dorigo and Gambardella (1997); Stützle and Hoos (1997). While population based heuristics have much in common with the system proposed in this thesis, there are a number of big differences. In population based systems there are many agents that are relatively simple normally performing some kind of crossover or perturbation heuristic move. The meta-heuristic agents of the system proposed in this thesis, are relatively few in number and execute more complex behaviours. Also population systems only use one meta-heuristic and are tuned to work on one problem at a time. This is different from the asynchronous generic modular approach taken in this work, where the agents are able to work on many different problems with little reconfiguring.

6.3 The formulation Symmetrical Travelling Salesman Problem

The Symmetrical Travelling Salesman Problem (STSP) can be formulated in the following manner. Given an n *by* n symmetric matrix of distances between n cities, the task for the symmetrical travelling salesman problem is finding the minimum length tour of all the cities, visiting them only once. A tour, or Hamilton cycle, is an undirected graph $G(V, E)$ where V is a set of vertices and E is set of edges. $|V| = n$ is

the number of vertices and σ_i is just a vertex in the i_{th} position in V $\sigma_i \in V$. (Reinelt, 1994)

An edge is just two vertices paired together $(\sigma_i, \sigma_{i+1}) \in E$ where the scripts are taken to be modulo n so $(\sigma_n \equiv \sigma_0)$.

A tour of unique list of cities is $S = (\sigma_1, \sigma_2, ..., \sigma_i, ...\sigma_n)$. The objective function value $Z(S)$ is the sum of all the lengths of its edges, thus is:

$$Z(S) = \sum_{i=1}^{n} d(\sigma_i, \sigma_{i+1}) \tag{6.1}$$

6.4 Computation experiments

Three different sets of computational experiments has been conducted on the STSP, PFSP and NR to test the effectiveness of the framework.

All computation experiments presented in this thesis have the following objectives The aim of the testing was to show the following:

a) show that cooperation produced better results than no cooperation

b) establish scalability by adding more agents improved results

c) understand if pattern matching protocol behaved as expected

d) establish whether simple heuristics cooperating can achieve good solution quality

e) establish if the system is flexible enough to work on different types of problem domains

f) establish if the framework can be extended to constrained problems

g) for the NR only establish that is possible to improve fairness by evaluating potential solutions with a new objective function.

6.4.1 Experimental settings

The platform essentially has the same configuration for each of the three case studies. The only differences concern the local search heuristic used with each problem and in the case of NR a third meta-heuristic (VNS) is used. This section describes the experimental settings common to all three of the case studies.

For the STSP and PFSP tests the platform is configured in such as way so that a meta-heuristic agent can either run Simulated Annealing (SA) or Tabu Search (TS) in conjunction with the local search heuristic.

All the case studies tests are each conducted with scenarios considering different groups of agents. These are:

Stand alone (1 meta-heuristic agent) TS, SA and VNS (nurse rostering only)

- 1 launcher agent
- 1 meta-heuristic agent running tabu search

4 Meta-heuristic agents

- 1 launcher agent
- 2 meta-heuristic agents running simulated annealing
- 2 meta-heuristic agents running tabu search

8 Meta-heuristic agents

- 1 launcher agent
- 4 meta-heuristic agents running simulated annealing
- 4 meta-heuristic agents running tabu search

12 Meta-heuristic agents

- 1 launcher agent

- 6 meta-heuristic agents running simulated annealing
- 6 meta-heuristic agents running tabu search

They were tested in this way so that standalone agents running just one meta-heuristic at a time could be compared with groups of cooperating agents to see which produced better results. Moreover different groups of agents were tested to see if adding more agents improved solution quality.

6.4.1.1 Parameter Settings

Each meta-heuristic agent can be configured from a file which is executed when the agents start. The agents were configured with different parameter settings described in the subsections below.

The first parameter setting is the number of conversations to be conducted in the course of a search. In the case of TSP and PFSP the agents each execute 3002 conversations, while for nurse rostering only 200 were required. Extensive testing showed that for these case studies these were the most appropriate number because after that no significant improvement in solution quality was gained. It was decided, therefore to choose these conversations limits.

6.4.1.2 Meta-heuristic settings

At start-up an agent can be instantiated to run one of the following meta-heuristics TS, SA and VNS. In the case of SA and TS each meta-heuristic evaluated its objective function 500 times. The TS tabu tenure was 7. VNS will be discussed in section 8.5.1.

For simulated annealing The geometric cooling schedule was used where the parameters were set at 0.9 so that meta-heuristic would diversify. As already explained in chapter V, this was to counter-balance the tabu-search agents which were configured with a small tabu-tenure set for intensification.

6.4.1.3 Performance measures for the evaluation of results

The results for each problem were averaged and the average percentage deviation from the known optimum was calculated. The percentage deviation from a known optimum was calculated using equation 6.2. Therefore, Table VI.4 shows the average percentage deviation from the known optima for these problems.

$$\frac{Method_{solution} - Best_{solution}}{Best_{solution}} \times 100 \tag{6.2}$$

6.5 STSP specific experimental settings

This section describes the experimental settings germane to STSP. The testing was conducted on the university network running a Linux Ubuntu 12.4 network using 13 student work stations. Each machine was a Hewlett Packard machine with an Intel dual core processor with 2 GB of RAM.

6.5.1 Local search heuristic

The local search algorithm used for these tests is a variant of the 2-opt algorithm (Croes, 1958) proposed by Beullens et al. (2003) with active marking and neighbour lists. The 2-opt algorithm was chosen for STSP because it is a well known local search heuristic for this problem. This variant finds good solutions more quickly than standard 2-opt algorithms reaching solutions of a similar quality. In all the tests using this algorithm the neighbourhood lists were all set to 1 so that each city has a full list of nearest neighbours.

6.6 STSP Benchmarks

The launcher agent, reads in a benchmark definition file. In the case of the TSP it reads in all of the benchmark files described below. It then executes a nearest neighbour greedy heuristic seed heuristic on each STSP problem read from file and sends each problem instance one at a time to the meta-heuristic agents to be improved.

Reinelt (1991) has produced a set of benchmark scenarios for all the different types of travelling salesman problem including a set of problems for the STSP. There are at least 50 problems in the STSP library alone and there are different types of problems with the STSP set depending on the coordinates provided for the position of a city. These coordinates can be given in 2 dimensions or 3 dimensions. Furthermore, the distance between cities can be calculated in different ways, such as, the euclidean distance, the manhattan distance, the maximum distance, the geographical and the pseudo-euclidean distance. For this work 14 STSP problems were chosen, all using the euclidean distance for 2 dimensions. The instances chosen are: a280(280 cities), berlin52(52 cities), bier127(127 cities), d198(198 cities), d493(493 cities), d657(657 cities), eil101(101 cities), eil51(51 cities), fl417(417 cities), gil262(262 cities), pr264(264 cities), pr299(299 cities), rat575(575 cities) and ts225(225 cities).

Each benchmark problem was run 10 times for groups of 12,8 and 4 cooperating agents where for each test the agents conducted 3002 conversations. In other words, each agent evaluated its objective function 3002 times. These are compared with non-cooperating stand alone agents which were run firstly executing simulated annealing and then tabu search.

The stand alone tests were conducted where the agent evaluated its meta-heuristic $3002 \times 500 \times 12 = 18012000$ times. This was done because each cooperating agent participated in 3002 conversations, each agent's meta-heuristic was executed at most 500 times and the largest group of agents was 12. Therefore overall the largest number evaluations of the objective function was done in the 12 agent group, 18012000 evaluations. To make a fair comparison, therefore it was necessary for the stand alone agents to perform the same number of iterations as the maximum overall iterations of the cooperating agents.

6.7 Results

It can be seen clearly from table VI.1 that there is quite a difference between agents cooperating and stand alone agents. The results show that cooperation outperforms

no cooperation. Although, the results of cooperative search cannot compete with the state-of-the-art heuristics that have been tailored to solve these benchmark problems to optimality, it can produce respectable results in a credible time limit. No search result took longer than 3 minutes. The aim of the framework is to build general domain-independent search methodologies that are capable of performing well across a wide range of optimisation problems and it is not expected to outperform meta-heuristics which are tailored for specific problems, but to give credible results. Table VI.1 establishes that the agents can produces results that quite close to optimality for the smaller problems. Indeed for eil51 the optimal results were achieved several times. It should also be noted that on eil51 the stand alone agent achieved better results that the 12 agents. This is because eil 51 is a very small example that can be easily solved to optimality by simple meta-heuristics. The agents conversation by pattern matching heuristic is a diversification strategy which does not work very well on small instances as the agents tend to diversify away from the good solutions provided by an agents meta-heuristic. To this end this diversification strategy is not suitable for very simple problems that can be solved easily.

It should also be noted that the stand alone agents produce the same results for each problem. It is because the standalone agents are configured in exactly the same way as the cooperating agents. In all tests all agents used the same deterministic 2-opt algorithm described in section 6.5.1. This coupled with a function that terminates a meta-heuristic search if the objective function value was repeated more than twice, means that in stand alone mode a search will always end up at the same local minimum. This is because such a sharp stopping criterion does not allow a TS aspiration function or SA objective function to accept worse moves. This always happened over a variety of testing programs conducted over a period of two years. Table VI.2 shows some earlier STSP tests conducted early in the PhD. The agents were configured differently but used the function for detecting local minima. The stand alone results are identical.

However table VI.3 conducted the same tests as in table VI.1 for stand alone

Table VI.1: The average percentage deviation from optimum/upper bound for each problem type

	12 Agents	8 Agents	4 Agents	1 Agent SA	1 Agent Tabu
eil51	0.56	0.47	0.75	**0.09**	0.09
berlin52	**6.94**	7.51	7.78	16.69	16.69
eil101	1.46	**1.40**	1.57	11.45	11.45
bier127	**2.35**	2.44	2.45	5.93	5.93
d198	6.28	6.63	**6.27**	12.50	12.50
ts225	**5.16**	5.29	5.44	15.76	15.76
gil262	12.93	**12.23**	13.12	27.92	27.92
pr264	8.22	**7.53**	7.95	16.25	16.25
a280	10.28	**9.65**	10.20	15.47	15.47
pr299	12.94	**11.96**	12.43	18.06	18.06
fl417	10.57	**9.96**	10.46	21.29	21.29
d493	**10.36**	10.71	10.84	16.94	16.94
rat575	12.22	12.60	**12.18**	18.41	18.41
d657	15.69	16.03	**14.68**	20.37	20.37

Table VI.2: Historic average percentage deviation tests presented here to show that the stand alone results are the same

	12 Agents	8 Agents	4 Agents	1 Agent SA	1 Agent Tabu
eil101	3.69	2.70	4.13	10.02	10.02
a280	13.57	14.17	13.06	15.82	15.82
gil262	14.83	14.85	15.29	15.43	15.43
pr299	15.80	16.29	17.33	18.06	18.06
rat575	18.18	18.35	18.65	18.81	18.81

agents. But this time the local minimum detection function had different settings for the standalone agents conducting the TS and SA tests. The SA stand alone agent had a local minimum detection function set to 100. This means that if the simulated annealing algorithm can continue with a search accepting worse values. It can be seen from table VI.3 that the resuts between the stand alone tests are sightly different. All other parameters remained the same.

Table VI.3: Standalone agents executing TS and SA where SA has a different local minimum detection criterion

Problem instance	Tabu	SA
a280	168508.6	169929.6
berlin52	38469.1	38364
bier127	143330.9	136298
d198	11279.7	10336
d493	14438.2	15408.1
d657	133588.4	133360.1
eil101	108977.2	109819.3
eil51	197581.4	212885.6
fl417	4646.3	4840.1
gil262	822.8	840.8
pr264	5127.1	5288.1
pr299	507.8	505
rat575	80264.7	74797.3
ts225	58006	61699.6

Figure 6.1 is a graph of boxplots validating the statistical significance of the observed differences, an analysis of variance (ANOVA) performed (Montgomery, 2008) with Tukey intervals. There is no overlap between cooperating and standalone agents showing that cooperation is better than no cooperation. This can also been seen in

table VI.1 where the final two columns show that the stand alone results are at least 5

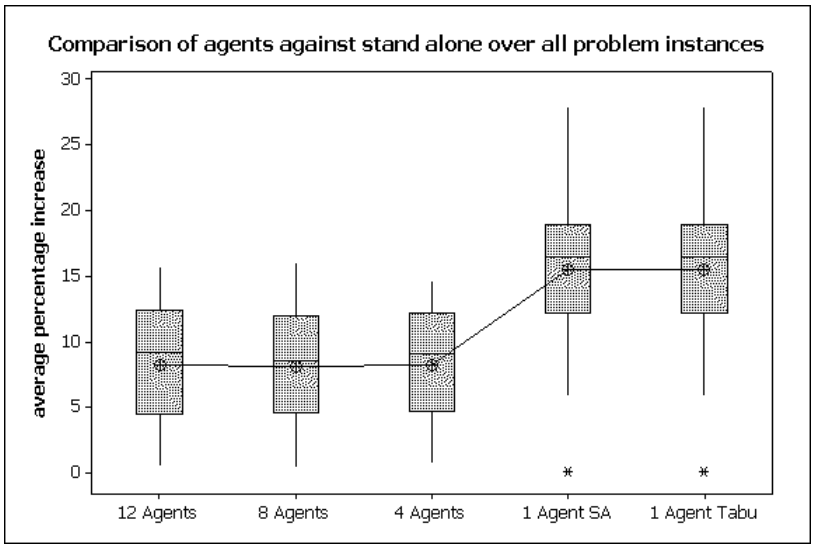


Figure 6.1: Cooperating agents versus no cooperation

A close study of Figure 6.2 and table VI.1 show that on average there is small improvement between 12 and 4 agents but the improvement is not significant. It shows that 12 and 8 agents have very similar results while 4 agents produce slightly worse results. This does not show conclusively that adding more agents improves results. It will be seen in chapters VII and VIII the trend for better results with more agents is more pronounced for PFSP and NR. However the results are suggestive of this trend rather than proof. To establish this claim further tests will need to be conducted with more agents and over longer testing epocs.

6.7.1 Patterns for robustness and diversification of the search

Tests were also conducted to understand how patterns were used during the search. These tests were conducted with groups of 4, 8 and 12 meta-heuristic agents. Tables VI.4 and VI.5 show the average number of patterns generated by these groups of meta-heuristic agents over the course of 10 iterations of the benchmark problems gil262(262 cities) and pr264(264 cities). These two examples were chosen because

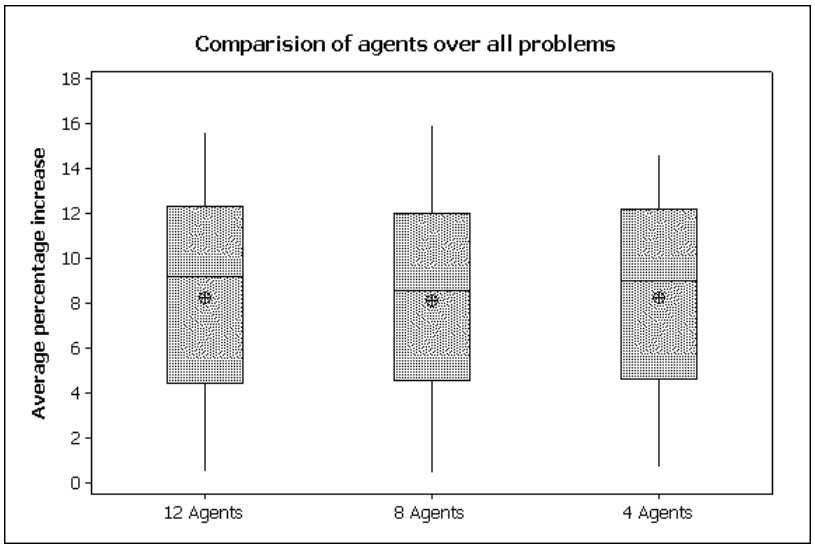


Figure 6.2: Comparison of result from 4 to 12 agents

they have roughly the same number of cites and yet they produce two the different pattern types. The patterns counted were only those that achieved good scores in the pattern matching algorithm. While in the case of pr264 it is clear that more patterns are generated by large groups of agents, it can be seen from gil262 that the 8 agent group produced on average less patterns than the 4 agents group. Extensive testing has shown that these patterns came in two types. In the first type the patterns were generated start of a search, usually within the first 3%. For the second type usually only 2 or patterns were identified as important by the agent but were constantly used throughout the search to generate new potential solutions. In this way large amounts of patterns are generated but they were often the same patterns being used in potential solutions.

Table VI.4 shows the number of patterns where patterns of type 1 predominated. For example run 1 of 4, 8 and 12 agents are of this type. However table VI.5 shows pattern numbers where type 2 was more prevalent. Run 6 of 4,8 and agents is a good example of this pattern type. It is important to note from testing so far it is not possible to predict which pattern type produced the best results.

This observation matches with the intended design of the pattern matching pro-

tocol as explained in 5.4.2.1. The idea is that pattern matching is used as a diversification technique within the protocol. The agents diversify by the exchange of good patterns in the first 3% of the conversation and then settle down to intensify generating no new patterns. On the other hand they use the same two or three patterns intensifying throughout the rest of the search. It is important to note that in any group of agents engaged on a search both types of patterns behaviour can occur.

Table VI.4: The average number of patterns for different groups of agents of 10 instances of the gil262 problem

gil262	run1	run2	run3	run4	run5	run6	run7	run8	run9	run10	average
4 Agents	33	121	27015	120	27019	26614	27094	82	94	27017	13520.9
8 Agents	150	16	145	772	38868	14695	16	214	22	16	5491.4
12 Agents	24	12141	35285	24	203	987	27630	24	76345	26	15268.9

Table VI.5: The average number of patterns for different groups of agents of 10 instances of the pr2624 problem

pr264	run1	run2	run3	run4	run5	run6	run7	run8	run9	run10	average
4 Agents	28106	4646	28716	29939	53	30018	28008	28359	89	2098	18003.2
8 Agents	118	51368	52086	16	52234	55981	54033	52174	51035	54033	42307.8
12 Agents	77603	78051	229	24	77846	75302	75067	24	75061	24	45923.1

Figure 6.3 is a line graph showing the progress of the search involving 4 agents for the problem eil51(51 cities). A different coloured line represents each one of the agents. At each data point, the Y value is the result of one iteration of an agent's meta-heuristic and is, therefore, the local minimum attained by each agent, while each X value is the objective function value achieved for the that agent during each of the 3002 conversations conducted by the agent in this search. The lower objective function values range from 428 -430 where 426 is the known optimum value for this problem. The search therefore ranges between 430 and 660. It can be seen that agent 3 varies the most in it ranges in values from near 430 to 660, while other agents such are much more compact. However it can mostly been seen that the agents search

in a range of about 150 in this search. This is in line with the same test on other problems where it can be seen that the pattern matching protocol diversifies the search for each agent but does not allow the diversification to be too large so that the search is effective and can find good results.

6.8 Conclusion

The tests results indeed show that cooperative search produces better results than stand alone. There is also some evidence that increasing the number of agents improves solution quality. Furthermore, the pattern matching protocol diversifies a search allowing it so examine new promising areas of the search space. These results are achieved with agents using very simple meta-heuristics and local search heuristics all configured with the same parameter settings. The only thing that changes in each test group is the number of agents. It has been established that cooperating agents using different meta-heuristics produce good results for the TSP. To further confirm generality of the framework the system was tested in the same configuration of the PFSP the results are presented in chapter VII.

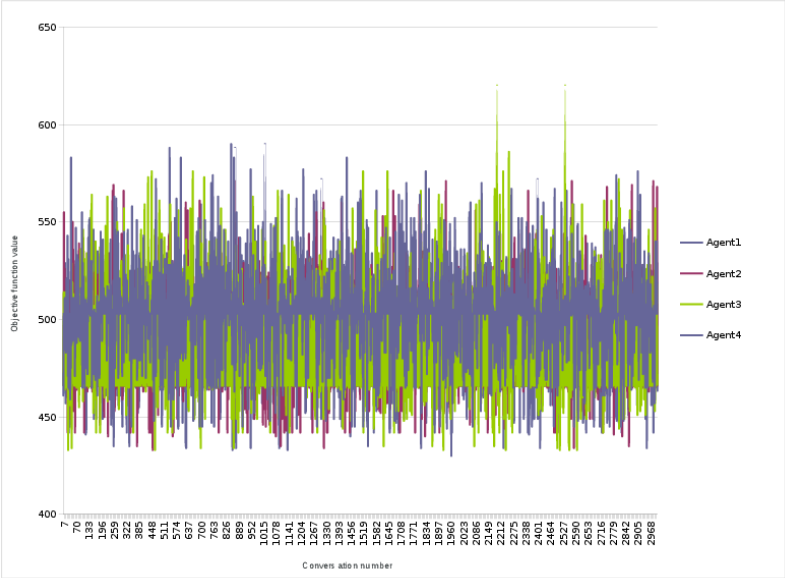


Figure 6.3: Comparison of result from 4 to 12 agents

CHAPTER VII
THE PERMUTATION FLOW-SHOP SCHEDULING PROBLEM

7.1 Introduction

The second of the case studies on the permutation flowshop scheduling problem is described in this chapter. Section 7.2 introduces the problem. The experimental settings are explained in section 7.3 and the results are analysed in section 7.4. Some conclusions are offered in section 7.5.

7.2 Permutation Flow shop Problem

The notation for describing this problem was first stated in two important early papers (Conway et al., 2003; Graham et al., 1979). The notation of Graham et al. (1979) (see above) is used more commonly these days. It consists of three distinct fields $\alpha/\beta/\gamma$ for describing the problem, as follows:

- the α field indicates the structure of the problem
- while the β field represents a set of explicit constraints
- and the γ field indicates the objective(s) to be optimized.

A complete description of possible values for the above mentioned fields was proposed by T'kindt et al. (2006).

The permutation flow-shop scheduling problem is a well known combinatorial optimisation problem which can be defined as follows. Given a set of n jobs, $J = \{1, ..., n\}$ to be processed on m machines, $M = \{1, ..., m\}$, where a job $j \in J$ requires a fixed non-negative processing time, denoted as $p_{j,i}$ on each machine $i \in M$, the objective is to minimise the *makespan*, $F|prmu|C_{max}$ or C_{max} (the completion time of the last job on the last machine) (Pinedo, 2002).

The completion time $C_{j,i}$ of job j on machine i is calculated using the following formulae:

$$C_{1,1} = p_{1,1} \tag{7.1}$$

$$C_{1,i} = C_{1,i-1} + p_{1,i}, \text{ where } i = 2, ..., m \tag{7.2}$$

$$C_{j,i} = max(C_{j,i-1}, C_{j-1,i}) + p_{j,i},$$
$$\text{where } i = 2, ..., m, \text{ and } j = 2, ..., n \tag{7.3}$$

$$C_{max} = C_{n,m} \tag{7.4}$$

There are $n!$ possible sequences and therefore the problem is known to be NP-complete (Chen and Bulfin, 1993). Many exact methods, heuristic and meta-heuristic approaches ranging from simulated annealing to genetic programming have been proposed for solving the permutation flow shop scheduling problem (Nawaz et al., 1983a; Ruiz and Maroto, 2005; Vázquez-Rodríguez and Ochoa, 2011). As far as is known the only work cooperative search research conducted on this problem is Vallada and Ruiz (2009). They proposed an island model where a genetic algorithms were used as islands which communicated whole solutions through a common pool.

7.3 Experimental settings

In these experiments each instance was tested using well known benchmark problems. To facilitate testing the cooperation protocol some simple (meta-)heuristics have been developed. The objectives of these experiments are the same as those in 6.4. It should be noted that the overall solution quality will be only as good as the (meta-)heuristics used. Success will be measured in the difference between performance between the same algorithms with or without cooperation.

The testing regime is exactly as that set out in section 6.4.1 "Framework experimental settings" of the previous chapter. This includes the parameter settings subsection 6.4.1.1, meta-heuristic settings subsection 6.4.1.2 and measurement of results subsection 6.4.1.3.

The objective of these tests is the minimisation of the total makespan.

The testing was conducted on the university network running a windows xp network with Novell NetObjects running on top using 13 student work stations. Each machine was a Dell 755 with an Intel dual core processor with 2 GB of RAM. The framework is coded in JAVA with 1 GB of RAM available to each agent.

Taillard (1993) has produced a set of 120 benchmark problems for PFSP which are known to be very hard to solve. Indeed, the optimum value has not been found for 33 of these problems. The problems are in various sizes 20, 50, 100, 200 and 500 jobs and 5, 10, 20 machines. There are 10 problems inside every set and in total there are 12 sets. These are 20x5, 20x10, 20x20, 50x5, 50x10, 50x20, 100x5, 100x10, 100x20, 200x10, 200x20, 500x20.

The tests were conducted using all of Taillard's problem instances where each one was executed 5 times. This was because each problem size has 10 different instances giving a comparable test sample to the ofter case studies in this thesis.

The local search algorithm used by all the (meta-)heuristic agents was a simple hill climber swapping two jobs. The jobs were chosen at random using a random number generator modulo the number of jobs. Each time this was done a pool of solutions was created which was half the length of the number of jobs. The best solution was chosen from this pool.

As has been explained, each problem instance was tested 5 times where each instance was tested with the system running 12, 8 and 4 agent scenarios. This was also applied to each stand alone scenario of simulated annealing and tabu search as described in section 6.4.1.

7.4 Test results

To compare these results with the best known upper bound or optimum, the 5 runs for each scenario, for each problem instance were averaged. The average percentage increase makespan over the upper bound was found according to the equation (6.2) above using the average for each problem instance. Next the average percentage increase for each problem type was calculated. This was done by taking the average

of the average percentage increases found for all 10 instances of a problem size.

Table VII.1 shows this average percentage increase for each problem size for each agent grouping. The "1 agent" values are given for when each agent ran SA ans TS as its (meta-)heuristic.

Table VII.1: The average percentage increase above optimum/upper bound for each problem type

	12 Agents	8 Agents	4 Agents	1 Agent SA	1 Agent Tabu
20x5	**0.12**	0.16	0.30	20.06	10.26
20x10	**0.81**	1.00	1.07	22.44	12.82
20x20	**0.90**	0.92	1.13	17.03	9.13
50x5	**0.09**	0.12	0.16	13.92	5.37
50x10	**1.38**	1.44	1.54	22.80	10.96
50x20	**3.32**	3.39	3.47	24.92	13.01
100x5	**0.14**	0.18	0.18	10.84	2.94
100x10	**0.67**	0.76	0.77	18.10	6.57
100x20	**2.53**	2.64	2.68	23.04	11.67
200x10	**0.48**	0.61	0.66	23.04	4.38
200x20	**2.24**	2.30	2.32	21.64	9.58
500x20	**1.56**	**1.56**	1.61	13.52	4.92

Table VII.1 shows clearly that there is a big difference between cooperating and non cooperating agents. It is also clear that as the number of agents increases, the results get better.

It is worth noting that 50x20 and 100x20 problems are especially hard where the optimum is not known. This can be seen especially in the results from 50x20 problem, the cooperating agents are between 3 and 3.5 percent above the upper bound while the stand alone agents range from just over 13 percent to near 25 percent above the upper bound. But even with these more challenging problems the benefits of

cooperation compared to no cooperation are clear.

However in the machine tests where there were only 5 machines, the system repeatedly found the optimum values. This was especially true of the 20x5 problem instances.

It should be noted that it took slightly longer for tests where there were 12 agents rather than 4 agents participating and this was not significantly longer. The smaller problems 20X5 and 20x10 took about 2.5 minutes to complete but that these times got progressively longer as the problem size increased. The 500X20 problems took about 8.5 hours to complete.

The spread of these results can be analysed more clearly using ANOVA performed with Tukey intervals (Montgomery, 2008). In figure 7.1 it can be seen that the cooperating agents have a clear difference in their performance from the stand alone agents. This can be seen as there is no overlap between the box-plots of the cooperating and stand alone agents.

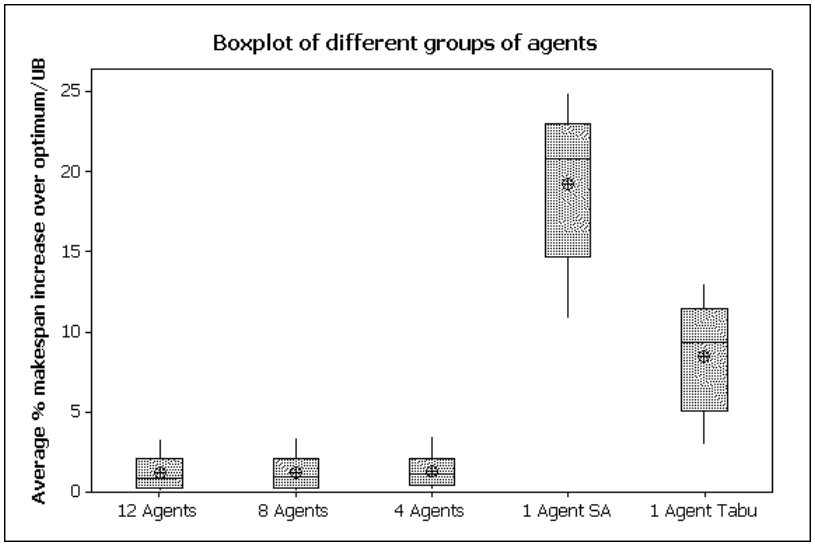


Figure 7.1: Cooperating agents versus stand alone

The second test objective mentioned in section 5.2 above was to see if more agents cooperating produced better results than fewer agents cooperating. Table VII.1 shows that while there is an overlap between the box plots of 12,8 and 4 agents cooperating,

it is clear from the this table that 12 agents perform better than 8 and 8 agents are better than 4. This can also be seen to some extent in Figure 7.2. There is a clear pattern emerging showing that more cooperating agents are better than fewer.

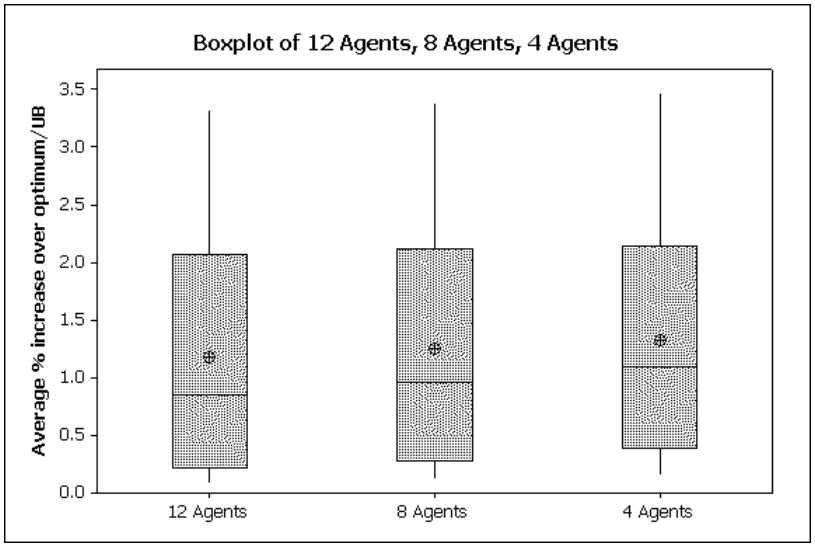
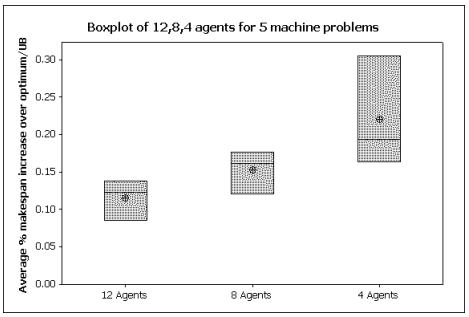


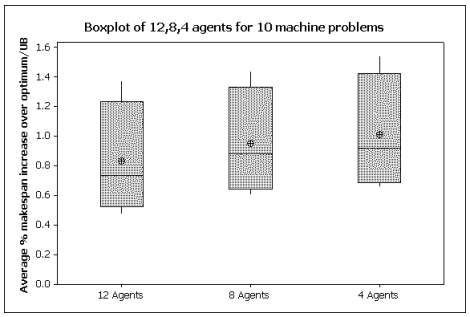
Figure 7.2: Comparison between 12,8 and 4 agents

It can also be seen from figure 7.3, that the best results for the cooperating agents are when the system tries to solve 5 machine problems to 20x5,50x5,100x5. But as the number of machines increases such as 20x10, 20x20, the system performs less well. This phenomenon also happens elsewhere in the literature. For example this can been seen in the results of Vallada and Ruiz (2009).

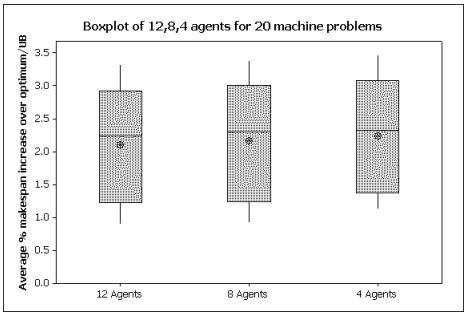
This phenomenon could be due to the fact that as the number of machines increases so does the possibility of longer times between the end of one job and the start of another or lag time. This could be explained by the calculation of the makespan (equation 7.4) taking the maximum between two jobs in different rows of the matrix of job times. Therefore, as the number of machines increase, the number of rows in the job times matrix increases and therefore the number of maximum functions increases. This in turn would lead to the possibility of greater lag times between the end of one job and the start of another. Consequentially, it would then make it harder to solve these types of problem as more possibilities have to be explored to



(a) 5 machines



(b) 10 machines



(c) 20 machines

Figure 7.3: Comparing agent performance on 5,10,20 machine problems

find a good permutation with a minimum makespan.

7.4.1 Diversification and robustness of the search by exchanging patterns

The system has a mechanism for recording how much of the search space was covered by the agents. This is helpful in understanding how much the search is diversified by exchanging patterns. This was achieved by getting each (meta-)heuristic agent to record the best value it achieved every time an agent's (meta-)heuristic was run. Each agent's meta-heuristic was run 500 times as set in the configuration file. However, the meta-heuristic agent has a mechanism to detect when a local minimum has been reached. To this end, the value recorded was either the detected local minimum or the best value calculated by the meta-heuristic after 500 iterations. There are 3002 values for each agent as there are 3002 conversations conducted during the course of the search.

Figure 7.4 is a line graph of a 4 agent scenario working on the first of Taillard's 10 20x5 PFSP problems. As this is a PFSP problem the values recorded on the Y-axis are the minimum makespan for each permutation of jobs found by each agent. The X-axis records the conversation number between 1 and 3002 conducted by the meta-heuristic agents. The different coloured lines represent the values recorded for each agent. In this way it is possible to compare the search path of each agent. This mechanism is completely generic and works on any problem type and with any conversation scenario.

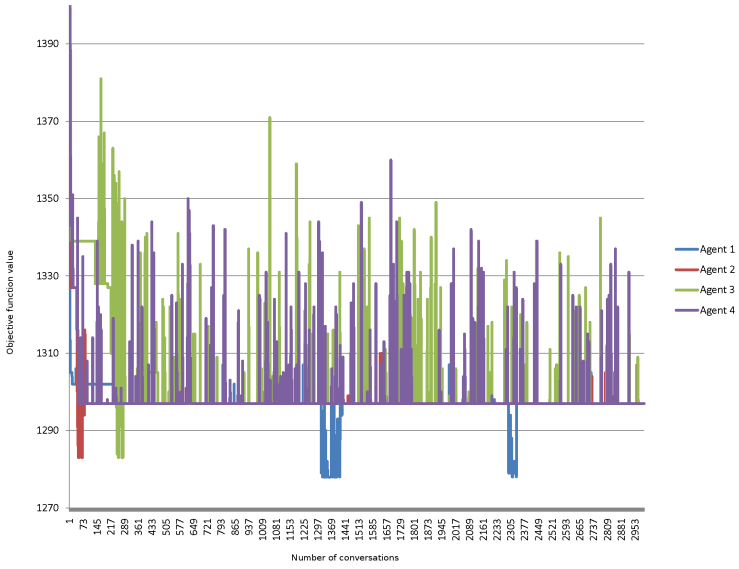


Figure 7.4: A graph of all the local minima or best results achieved by each agent

What is clear from figure 7.4 is that while the agents do traverse different parts of the search space at the start of the search, they quickly converge to a good solution which is also a local minimum (in this case 1297). Meta-heuristic agent 1 is the only agent that reaches the optimum 1278, which it does twice in the course of the search. Meta-heuristic agents 2 and 3 achieve good solutions early on, but then get pulled back to the 1297 local minimum. Meta-heuristic agent 4 starts from the worst solution of all and gets down only as far as 1297.

One inference that can be drawn from this is that while pattern matching enabled

meta-heuristic agents 1,2,3 to get into a position to achieve the good solutions early on in the search, Meta-heuristic agent 4, which started off worse than the others, seems have converged to the local minimum 1297. And from there it seems to have influenced the search of the other meta-heuristic agents by sharing patterns that lead it to converge to 1297, holding back the other meta-heuristic agents from achieving better solutions. However, the reinforcement mechanism ensured that the meta-heuristic agents were never pulled to far away from the optimum.

The information collected in table VII.2 shows how many patterns the agents identify as potentially good patterns for any new solution. It can be seen that, as might be expected, as the number of agents increase the number of patterns increases, but as with the STSP there is quite a lot of variation between problem instances. Some runs generate more patterns than others. Furthermore, on different tests of the same problem instances there can be big differences between the numbers of patterns identified by the agents. This variability is to be expected as no search using this type of distributed algorithm (see chapter IV) is deterministic. However as has already been mentioned in section 6.7.1 the patterns always occur at the beginning of a search or the same small number of patterns are used by an agent throughout the search. In each case this lends weight to the conclusion that the pattern matching protocol diversifies the search after each conversation just enough to cause the agents to search promising areas of the search space.

Table VII.2: The average number of patterns for different groups of agents of 10 instances of 20x5 problems

Agents	run1	run2	run3	run4	run5	run6	run7	run8	run9	run10	average
4 Agents	10	217	459	416	46	27	516	76	96	29	189.2
8 Agents	202	244	220	16	361	86	939	190	49	260	256.7
12 Agents	681	2445	346	80	708	499	764	819	456	1684	848.2

7.5 Conclusion

At the beginning of this chapter (see section 6.4), a number of test objectives were set. By referring back to these objectives, it can be seen that the system has performed well. With respect to objective a) the system has been tested very successfully on STSP and PFSP. Furthermore, as required by objective e), the system is in exactly the same configuration as the system was for STSP, the only difference was the local search heuristic used. Yet the the system produced good results for both types of problem. Indeed the system has been particularly successful producing optimal and near optimal results for many of the benchmarks in PFSP. Another objective (b) was to show that agents cooperating is better than stand alone meta-heuristics. Again, the system has outperformed the stand alone agents showing a clear solution quality advantage. With respect to objective d), the pattern matching protocol behaved as expected, allowing the agents to diversify just enough to cause the agents to search promising areas of the search space. Finally it is clearer for PFSP than for STSP that more agents working on a problem increases the solution quality produced by the system as required by objective c).

CHAPTER VIII
THE NURSE ROSTERING PROBLEM

8.1 Introduction

To test the flexibility of the system, It was decided to test the system should not only be tested on a constrained problem but a new model as well. To this end the system was adapted to work on problems in fairness in Nurse rostering problem. A modified version of model of Bilgin et al. (2012) was implemented. The resulting system was tested on a modified versions of the well known real world benchmark problems developed by Bilgin et al. (2012). The new fairness model represented nurses preferences as personal contractual constraints. A new fairness objective function was different to the standard sum of nurse roster violations in order to focus on an even distribution of violations across nurse rosters. The aim here is to show that each nurse's preferences as violations are treated fairly. In this way the framework and system could be shown to be able to solve constrained combinatorial optimisation problems as well as new models.

This work shows that by the use of of ontologies the framework and system are generic and modular making it easy to implement a new model and adapt the system with a new heuristic and objective function. Again it was shown that the agent-based cooperation outperforms the more traditional stand alone approach. But the system also produced fairer nurse rosters. Furthermore using the standard sum of violations, the system achieved impressive results as well. The system is therefore generic across different problems but can also be adapted for new models.

The nurse rostering problem and its background are discussed in sections 8.3 and 8.6. The model of Bilgin et al. (2012) and its adaptation using the framework is introduced in sections 8.3 and 8.4. The experimental design is discussed in section 8.5.1. The results for the first bank of tests with the tradition sum of roster violations model are discussed in Section 8.5. While section 8.6.1 introduces the experiments and results for the new fairness model. Finally section 8.7 concludes the chapter with

a discussion on the contribution of this work and some directions for future research.

8.2 Background

The health care sector is under increasing pressure due to the ageing population and increasing cost of ever improving treatments (Rais and Viana, 2011). Moreover, many health care organisations suffer from a shortage of nursing staff. Nurses are responsible for the majority of health care duties and experience a lot of stress on a daily basis. Job dissatisfaction appears to play a key role in the high resignation rates of nurses. Mueller and McCloskey (1990) identified 'scheduling' as the second most important factor contributing to job satisfaction, after salary, vacation and other work benefits. Improving the schedules of nurses is therefore a sensible way to increase their satisfaction and consequently staff retention (Larrabee et al., 2007).

The task of building a personal schedule for each nurse in a hospital is known as nurse rostering in the literature and is often classified as a timetabling problem. The problem usually considers assigning nurses to a set of shifts in such a way that required shifts are covered by nurses with the best possible skill match. Besides this requirement, the resulting assignment should optimise constraints on the nurses' individual rosters to maximise the quality of their work life balance. Burke et al. (2004b) present an overview of constraints and objectives that are common to many nurse rostering problems. This survey, in addition, classifies many mathematical as well as heuristic approaches that have been applied to different variants of the nurse rostering problem.

The work undertaken on fairness reports on a methodology for increasing the nurses' satisfaction with respect to their personal roster. Automatically generated rosters are commonly evaluated by means of a weighted sum objective function, the result of which is proportional to the number of soft constraint violations (Burke et al., 2001b). However, such approaches do not necessarily compare well with the human way of assessing the quality of a roster. Two rosters with the same objective function value may differ considerably in terms of a pairwise comparison of individual roster's

quality. Beddoe and Petrovic (2006) apply a completely different methodology for quality assessment, namely case-based reasoning. The approach requires a training phase in which experts, i.e. nurses, are invited to identify poor elements and to modify the roster such that the problems are sorted out. Objective function based approaches and case-based reasoning are hard to compare in a quantitative manner (Petrovic and Berghe, 2008).

Modelling mathematically the perceived quality of a roster by individual nurses would definitely be much harder than the weighted sum approach. Such a model is likely to be blurred by individual time-varying nonlinear dependencies between constraint violations and by the rosters that have been obtained for the fellow nurses. Therefore, some assumptions and simplifications are inevitable.

The aims of this chapter are to show how the framework can be easily adapted to a constrained problems such as nurse rostering thorough the use of the ontology described in chapter V. Furthermore this is easy to implement because of modular design of the system as explained in section 8.4. Testing will show that the system produces results as good as the state of the art.

Another aim of this chapter is to propose a set of new evaluation models for nurse rostering, which capture the concept of fairness of work better than the existing models and hence optimise the rosters accordingly. Different fairness measures are introduced and their attainability is investigated experimentally. Testing is undertaken using data sets that were obtained from a hospital in Belgium and apply stand alone and cooperative meta-heuristic approaches to generate fair nurse rosters. Apart from a number of conference presentations, very little publications are available. Wang and Wang (2009) developed an agent-based approach to self rostering. Haspeslagh et al. (2009a) addressed the problem of exchanging nurses between wards and sorting out personnel shortages. Haspeslagh et al. (2009a) present a Pareto optimal negotiation approach for assigning the workload across different wards. As far is known, there is scarce research work on fairness and cooperative search in nurse rostering.

8.3 Modelling of the nurse rostering problem

Most real-world nurse rostering problems can be represented as constraint optimisation problems using 5-tuples $\langle N, D, S, K, C \rangle$:

N: Set of nurses

D: Set of days in the current schedule period and in the related parts of the previous and coming schedule period

S: Set of shift types

K: Set of skill types

C: Set of constraints.

$x_{n,d,s,k}$ denote the decision variables and $\forall n \in N, \forall d \in D, \forall s \in S, \forall k \in K$:

$$x_{n,d,s,k} = \begin{cases} 1 \text{ if employee } n \text{ is assigned to shift } s \text{ and skill } k \text{ at day } d \\ 0 \text{ otherwise} \end{cases}$$

Where the 4-tuples $n, d, s, k \in mathbbN$ is the assignment.

The nurse rostering problem can be defined as an integer program. Let N *be a set of nurses and* T be a set of tasks representing the requirements of shift a on specific day needing certain skills. To this end $D, S, K, C \in T$.

Also let P be a penalty function for violating soft constraints if nurse i is assigned to task j defined as:

$$P(i, j) \in \mathbb{R} \; \forall i \in N \text{ and } j \in T \tag{8.1}$$

The nurse rostering problem can be formulated as an integer program as follows.

The objective function is defined as:

$$\min \sum_{i \in N} \sum_{j \in T} P(i, j) x_{i,j} \tag{8.2}$$

Subject to the following constraints:

$$\sum_{j \in T} x_{i,j} \geq 1 \text{ where } i \in N \; and \; j \in T \tag{8.3}$$

The variable x_{ij} represents the assignment of nurse i to task j. The constraint requires that every nurse is assigned to at least one task.

The assignment of values to the decision variables is strongly restricted by constraints. The literature presents various ways to dealing with the constraints by, for example, considering some as hard and some as soft constraints. The former need to be satisfied in order to produce a feasible solution, while the latter need to be satisfied as much as possible in order to find high quality solutions. Most problems are very complex and it is usually impossible to generate solutions satisfying all soft constraints.

Common models distinguish between coverage constraints and time related constraints. The first category includes constraints limiting the difference between required nurses on a shift, skill, day and nurses actually assigned to this shift, skill, day.

In contrast, time related constraints restrict the assignments within nurses' timetables. The class of time related constraints includes on one hand contractual constraints covering all the permanent rules such as minimum/maximum working time and on the other, minimum/maximum consecutive assignments to shifts, days, and so on. Contractual constraints are typically grouped into full time nurses, weekend workers and many different part time contracts (30%, 75%, 90%, or never on Wednesday afternoon). It should be clear that the possible contract definitions are endless. On the other hand, personal requests for a day on/off or for another short leave also belong to this class. Some particular time related constraints are noteworthy in a fairness context. Balancing working hours or weekend work among full time nurses are examples of fairness related constraints.

In the present model, the following are hard constraints:

- A solution is feasible when only a nurse can be assigned to one shift each day.
- No assignments are allowed for which a nurse is not qualified.
- Assignments which occur on two consecutive days but one after each other are

not allowed.

- Assignments are only considered feasible if they are defined in the coverage constraints

This last constraint is motivated by the fact that in practice there are assignments which will never occur, such as a head nurse will never be assigned to a night shift. No such coverage constraints will be defined, and by imposing this constraint as a hard constraint, such assignments will never be made in a solution.

There are also a number of soft constraints which while a potential solution remain be feasible, violation will result in a penalty.

(1) Coverage constraints are considered soft constraints. This allows for assigning nurses to a particular shift, skill, day, even when the maximum has been reached. In practice, this is sometimes necessary in order to meet the nurses' required working time.

(2) Another soft constraint is the minimum required rest time between two consecutive shifts, which is typically set to 11 hours.

(3) Multi-skilled nurses can be defined in the present model. Typically, this is used to model primary and secondary skills by assigning weights to the different skill types. Assignments in which a nurse uses a secondary skill are feasible, but will incur a penalty for doing so.

(4) The aforementioned time related constraints can be further categorised into three types of soft constraints: counters, series and successive series (Bilgin et al., 2012).

(5) Counters are used to limit the occurrence of a specific subject in a particular period. Examples of counter constraints are no night shifts in a weekend or maximum 36 hours worked each week.

(6) Series restrict the successive occurrence of a specific subject in the scheduling period, e.g. minimum 3 and maximum 5 consecutive night shifts or no isolated idle days.

(7) Finally, successive series are used to restrict the occurrence of two consecutive series. Successive series are used to model constraints such as no early shift after a late shift or a series of night shifts have to be followed by at least 2 free days.

Many previously proposed approaches often define a weighted sum objective function WO to evaluate the quality of a roster. It is defined as follows:

$$WO = \min \sum_{i \in N} \sum_{j \in T} P(i,j) x_{i,j} \tag{8.4}$$

The agent-based system is tested using this objective function and the results are compared with those of Bilgin et al. (2012).

Several meta-heuristics have been successfully used to solve the nurse rostering problem (Edmund et al., 2004). However, to obtain the best performance from a meta-heuristic a lot of experimentation is required to determine the best parameter settings. Section 8.4 explains how the agent-based system has been developed to work nurse rostering problems. The nurse rostering system is a module of the main agent based system so that any agent executes its given meta-heuristic and generates new potential solutions that are validated against the constraints of the problem.

8.4 Implementation of the Nurse Rostering Model in the framework

The nurse rostering model developed by Bilgin et al. (2012) was implemented in Java and added to the platform. This was accomplished by making the Assignments class of the model a subclass of the SolutionElements abstract class of the platform (see figure 5.4). The Assignments class is where nurses of the correct skill type are assigned to shift to satisfy the requirements of a nurse rostering problem. In this way they form the basic elements of any potential roster which might be a solution to a given nurse rostering problem.

The SolutionElements class as has been explained in 5.4.1.1 is the basic element of any potential solution in a Combinatorial Optimization problem. These therefore, are the interfaces between any concrete problem and the generic agent-based system.

As such they are the obvious superclass of the Assignments objects as they are the building blocks of any potential valid roster. In this way the system was able to interface with the nurse rostering module quite simply, in that the Assignments objects, as they were generated by the module were given unique ID's and stored in memory. The agent-based system used its meta-heuristics to optimise potential solutions of ID's as explained in 5.4.2. These were then translated back into assignments object and passed to the nurse rostering module for verification against the constraints.

The model also uses the Variable Neighbourhood Search (VNS) meta-heuristic and the local search heuristics developed by Bilgin et al. (2012). These were implemented in Java and added to the platform. This is described in detail in section 5.3.

8.5 Experiments

The NR experiments have been conducted as explained in section 6.4. The final two additional objectives are to test the framework on a highly constrained problem (f) in section 6.4) and to use a new objective function with the aim of producing fairer nurse rosters (g) in section 6.4).

8.5.1 Experimental settings

Experiments have been conducted to evaluate the performance of the agent-based cooperative meta-heuristic approach compared with the stand alone approaches.

The experiments have been carried out using four different scenarios. These scenarios are based on existing wards in a Belgian hospital: emergency, geriatrics, psychiatry and reception (Bilgin, 2008). Table VIII.1 gives an overview of the instance characteristics. For each ward, two cases are considered: one where all the nurses have the same contract, and one where each nurse has an individual contract with both common and personalised constraints. The amount of constraints defined in each contract greatly differs between the instances. For example, nurses in the geriatrics ward are only subject to two constraints (limiting working hours and the number of consecutive days worked). In the psychiatry ward on the other hand, a large number

Instance	Nr of nurses	Nr of shifts	Nr of skills	Planning period
Emergency	27	27	4	28 days
Geriatrics	21	9	2	28 days
Psychiatry	19	14	3	31 days
Reception	19	19	4	42 days

Table VIII.1: Instance characteristics

of constraints are specified in each contract, restricting working time as well as specific patterns. In the cases where for each nurse an individual contract is defined, again the number of specified constraints differs greatly. Some of these constraints will be personalised, but there still exist a number of common constraints which apply to all nurses in the ward. For both cases, nurses follow one contract during the scheduling period, i.e. they do not change contract types during this period.

The meta-heuristic agents implemented in the framework to solve fairness in nurse rostering are described in section 5.3.

The experimental setting are exactly the same for the basic set up of the system and test testing scenarios used as described in section 6.4.1.

The experiments were first conducted using the standard WO objective function. The results of these experiments are given in section 8.5.2. In section 8.6.2 experiments were conducted using a new objective function modelling fairness. These results are also compared with the WO results.

As described in section 6.4.1 The tests are designed to compare different groups of agents with their stand alone counterparts.

For each scenario, the experiments were conducted over 10 runs for each problem instance. These experiments were run using the WO objective function. The agents conducted only 200 conversations to complete each search taking no longer than 6 minutes to produce a new roster for each problem instance. Each agent ran on a HP

Compaq 6000 pro with Intel Core 2 duo E8400 processor with 4 gigabytes of RAM in a 2x2 gigbyte configuration. The agents were configured to use only 1 gigabyte of memory.

The launcher agent reads in the problems and executes a greedy heuristic seed algorithm. The launcher then sends the seed and the problem definition to each of the meta-heuristic agents to solve. When the search is complete the launcher takes the best result from the meta-heuristic agents and outputs an XML file with the final roster and its objective value.

The meta-heuristic agents are configured in exactly the same way as the other case studies (see section 6.4.1.2).The only difference was that the a VNS agent was added.

In the case of stand alone mode, a meta-heuristic agent executes a given meta-heuristic for $200 \times 12 \times 500 = 1200000$ iterations, where 12 is the number of agents involved in the 12 agents network, 200 is the number of conversations and 500 is the number of times the agents meta-heuristic is executed per conversation. This is used to produce the greatest number of iterations used by any of the agent networks.

The number of conversations 200 is chosen because experimentation shows that the rate of solution improvement is reduced after that number. The 200 conversations lasts no longer than about 6 minutes and this is deemed to be a good stopping point.

8.5.2 Experimental results for the WO objective function

Table VIII.2 shows the overall average results respectively over 10 runs for each of the different scenarios. In each case the agents tackle the problems in groups of 12,8 and 4 and these are compared with the standalone case for each meta-heuristic used. Furthermore the results of Bilgin et al. (2012) are given as a comparison and show that the agent-based approach produces better results. Finally where known, the values for each roster of the human planner are given, these are also taken from Bilgin et al. (2012). While it is clear the human planner gets the best results, the agent-based approach gets best of all the algorithmic approaches.

Instance	12 Agents	8 Agents	4 Agents	Tabu	SA	VNS
Emergency-i	104,796.00	103,900.50	102,212.50	191,330.50	193,745.00	191,301.50
Emergency-d	103,534.00	105,705.00	112,119.50	148,844.00	161,888.50	150,685.50
Geriatrics-i	106,484.50	107,112.00	107,469.00	172,607.50	174,051.50	170,278.00
Geriatrics-d	104,640.00	103,702.00	105,234.50	175,540.50	174,370.50	164,053.50
Psychiatry-i	103,705.00	103,455.00	105,342.00	190,893.50	209,846.50	186,482.00
Psychiatry-d	105,330.00	107,597.00	108,878.50	160,882.50	154,293.50	146,529.00
Reception-i	105,015.00	103,532.50	105,071.50	172,759.00	181,258.50	176,850.00
Reception-d	104,727.50	106,520.00	112,210.00	174,476.00	170,824.50	170,170.00

Table VIII.2: Results for WO function for 12,8,4 agents and stand alone meta-heuristics running 3 meta-heuristics

To validate the statistical significance of the observed differences, an analysis of variance (ANOVA) has been conducted (Montgomery, 2008) with Tukey intervals. Fig 8.1 shows the ANOVA box plots. It shows the average results for agents calculating each problem instance over 20 runs. The different groups of agents solve the same problem and are compared with the stand alone agent running each meta-heuristic in turn to represent the case where a meta-heuristic is used with no cooperation. The results clearly show that there is clear gap between the stand alone approach and cooperative search. Also the distribution of results is much tighter for cooperative search than the stand alone approach.

8.6 Fairness in nurse rostering

A recent survey on operational research in health care points at the importance of optimising resource planning and scheduling (Rais and Viana, 2011). It is noteworthy that among the extensive number of papers discussed in that review, only a few pay attention to fairness of work distribution. Felici and Gentile (2004), for example, assume that the extent to which the contractual constraints are met is correlated with the nurse's satisfaction. However, the objective function sums 'satisfaction' over all the roster elements, thereby making no distinction between individual nurses' overall satisfaction. Some work schedules, especially those involving shift work, induce high

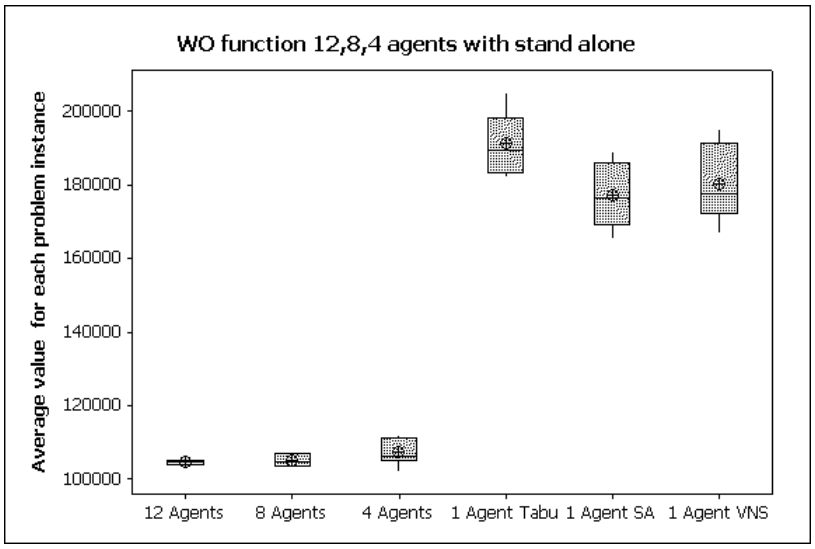


Figure 8.1: Comparison of Agents and stand alone calculating WO function averaged overall instances.

levels of fatigue. Yuan et al. (2011) present recommendations on shift sequences, days off and overtime so as to reduce the risk of fatigue as much as possible.

One of the most informative overview papers on nurse rostering (Warner, 1976) puts forward fairness as a quality measure. Warner indicates that an even distribution of work has a considerable advantage over cyclic schedules, despite their limited flexibility. Burke et al. (2004b) noticed also that the majority of nurse rostering papers do not explicitly address fairness. Approaches paying attention to this concern tend to model fairness as a balance constraint on working time accounts, while solving the problem with an optimisation algorithm (Burke et al., 2006; Meyer auf'm Hofe, 2001b). It is noteworthy that many recent nurse rostering papers take some work balancing measures into consideration (Causmaecker and Berghe, 2011).

Kellogg and Walczak (2007) investigated why only a small number of automated nurse rostering approaches are being used as decision support systems in hospitals. They pointed out, among some other causes, that academic models fail to meet the complex needs that health care organisations face. Interactive rostering, also referred to as self scheduling, is a mostly manual mode of operation that has gained attention

particularly because it potentially increases nurses' satisfaction (Robb et al., 2003). The nurses put together a roster by expressing their preferences and negotiating until the hospital's requirements are met. One major drawback of manual self scheduling is that the quality of the result depends on the nurses' ability to cooperate and negotiate, and therefore Rönnberg and Larsson (2010) advocate an automated approach.

They enforce some degree of fairness by means of an auxiliary variable representing the requests by the least favoured nurse, which was included in the objective to be maximised.

Grano et al. (2009) combine aspects from self rostering and from common optimisation methods into a two-stage approach. First, an auction is set up in which nurses can spend a number of points to bid for preferred shifts or days off. After the best bid assignments have been made, the remaining part of the problem is solved with a mathematical solver so as to meet the ward's staffing demands. This hybrid approach was tested on data sets from a real hospital ward but without active involvement of the nurses. The bids were derived from the nurses' preferences instead. Grano et al. (2009) indicated issues that require further research. Knowledge on popular shifts, for example, may influence a nurse to set a bid. In addition, fairness requires that points of bids not granted should be transferable to the next scheduling period. The next section 8.6.1) attempts to sort out such unfairness issues.

8.6.1 New model considering fairness

Despite its simplicity, WO includes some weaknesses when considering fairness. The function does not allow for distinguishing solutions with the same objective but composed of unbalanced violations with respect to the individuals. Therefore, it is worth considering a different objective function FO equation 8.5, which is equal to the maximum weighted sum of violations in an individual's roster. Again, FO should be minimised. Good values for FO are expected to correspond to rosters that are more fair than those obtained when optimising WO.

$$FO = min(max_{\forall i \in N}(\sum_{\forall j \in T} P(i,j)x_{i,j})) \tag{8.5}$$

8.6.2 Experimental results on fairness using FO objective function

For these experiments the agents are set up exactly as described in section 8.5.1. However these experiments were run using the FO objective function.

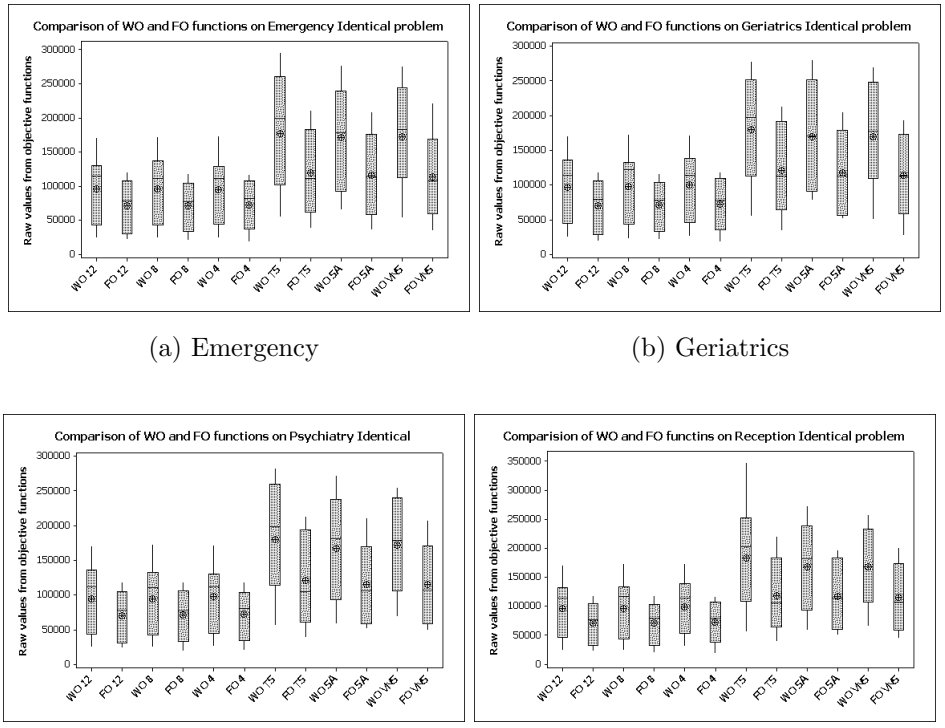
Instance	12 Agents	8 Agents	4 Agents	Tabu	SA	VNS
Emergency-i	77,864.50	77,627.50	79,507.50	138,761.50	138,761.50	122,709.50
Emergency-d	76,480.00	77,449.00	79,349.50	108,328.00	108,328.00	100,426.00
Geriatrics-i	78,125.00	78,012.00	80,065.50	115,394.00	115,394.00	114,948.00
Geriatrics-d	77,451.50	78,382.50	78,993.00	109,664.00	109,664.00	108,078.50
Psychiatry-i	77,561.00	77,972.50	79,863.50	130,234.50	130,234.50	121,514.00
Psychiatry-d	77,747.00	77,652.00	78,960.50	105,739.50	105,739.50	95,764.50
Reception-i	77,799.50	78,048.00	78,599.00	122,855.50	122,855.50	117,683.00
Reception-d	77,212.00	77,729.50	79,956.50	105,741.00	105,741.00	103,058.50

Table VIII.3: Results of averages for each problem for FO function for 12,8,4 agents and stand alone meta-heuristics

Table VIII.3 show the average over 10 runs of 12,8 and 4 agents cooperating for each of the fairness nurse rostering instances. The table also shows the results where a stand alone agent executes in turn each of the meta-heuristics used by the platform on the fairness nurse rostering instances. It can be clearly seen that agents cooperating produce fewer violations that stand alone agents. To validate the statistical significance of the observed differences, an analysis of variance (ANOVA) (Montgomery 2000) was performed with Tukey intervals is performed. Fig 8.2, show the ANOVA box plots. The figure shows the results for the *identical contracts* problems.

The box plots are grouped in two's from left to right 12 agents calculating WO

and 12 agents calculating FO through to the stand alone agents each running one of the three meta-heuristics. The results clearly show that there is clear gap between the stand alone approach and cooperative search. Also the distribution of results is much tighter for the FO search than the WO. This indicates that because of the even distribution of violations over the nurse rosters that these are fairer than the the WO rosters.



(a) Emergency

(b) Geriatrics

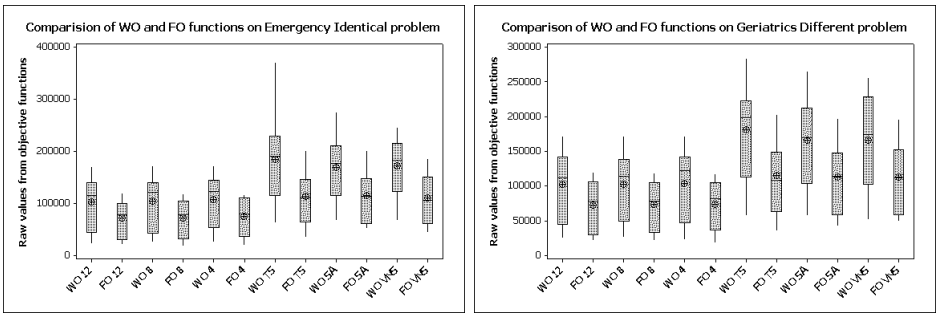
(c) Psychiatry

(d) Reception

Figure 8.2: Distribution of fairness in case of identical contracts

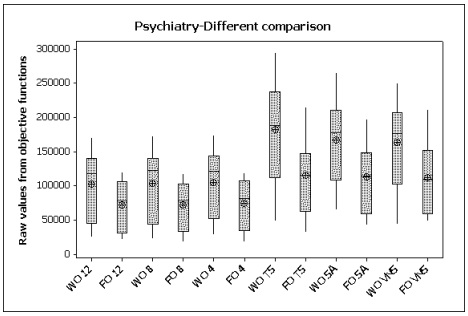
Fig 8.3, shows the results for the *different contracts* case. Again they show a similar trend that FO is a more effective objective function than WO in generating fairer rosters. Furthermore the cooperating search outperforms the stand alone approach.

These results show that cooperating agents produce is more effective results than their stand alone equivalents. The results clearly show that cooperative search that combines the strength of a variety of meta-heuristics outperforms the performance of a single meta-heuristic. In each case there is a clear gap between cooperative search

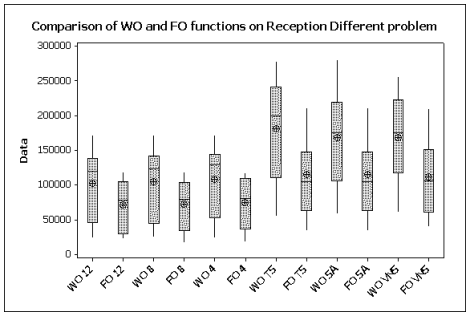


(a) Emergency

(b) Geriatrics



(c) Psychiatry



(d) Reception

Figure 8.3: Distribution of fairness in case of different contracts

and stand alone results. It is also noticeable that as the number of agents increases the overall roster quality is improved. This tends to show that adding more agents to solve a problem increases the solution quality which as was explained earlier is the reason for testing with different sizes of agent network. However more work will have to be done to substantiate this conclusion.

This work also shows that the framework can easily be extended to new models such as fairness in nurse rorstering. Here the results show that the new fairness model using FO produced results that had more even distributions of violations across the nurse rosters then the WO model. Furthermore, this result achieved amongst all the test instances. Furthermore cooperating agents executing this model also outperformed their standalone equivalents.

8.7 Conclusion

In this chapter the agent-based system was augmented with a model developed from the work of Bilgin et al. (2012) for solving problems in nurse rostering (Ouelhadj et al., 2012). This work was undertaken to show that the framework is modular and that the ontology was robust enough to be used to solve contained problems in combinatorial optimisation. The system was also used to investigate a set of new fairness measures as new objective functions of the nurse rostering problem.

Experiments were conducted using cooperative meta-heuristic search with the standard WO function achieving better results than the stand alone alternative.

Experiments were also conducted to explicitly model fairness into the objective function using FO. The resulting rosters proved to be fairer without deteriorating the quality with respect to other constraints. Furthermore, the results showed that cooperative search has generated the fairest rosters compared to the standalone approach. This conclusion should be adopted in future nurse rostering work because the modelling effort is limited and the complexity of the problem is not affected.

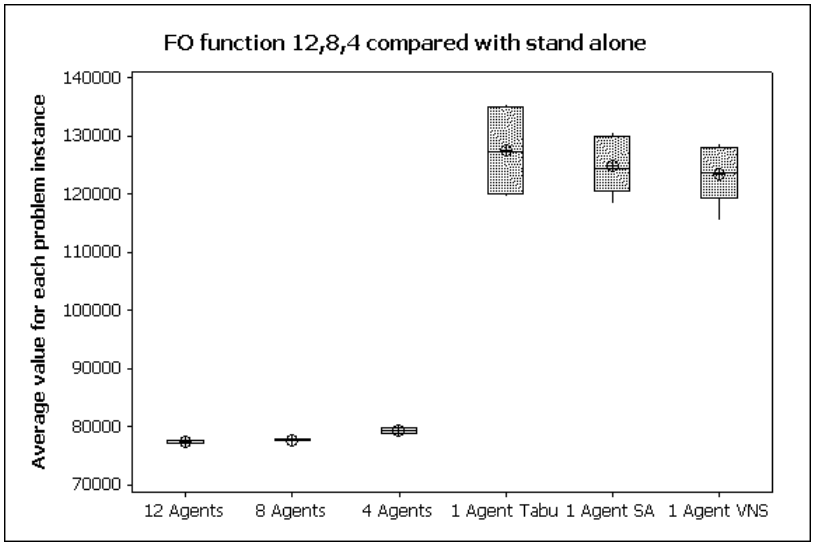


Figure 8.4: Comparison of Agents and stand alone calculating overall FO function

CHAPTER IX
CONCLUSIONS

It has been the aim of this thesis to show that there are two gaps in the operational research literature where: a) there has been little research into cooperating algorithms as Turing machines (IV), b) no research has been conducted on direct, autonomous cooperating agents instantiating different meta-heuristics (chapter II). These are addressed by the development of a generic, flexible, scalable and modular agent-based framework which can solve different types of combinatorial optimisation problems with little problem specific configuration. Also a new formal theory of cooperating Turing machines has been proposed to show they can execute a new type of algorithm that standard Turing machines cannot embody.

Chapter IV discusses formal representations of algorithms and proposes a system of Cooperating Non-deterministic Turing Machines (CNTM) with a choice transition relation. It also discussed more physical representations of agent-based system that exhibit the property of unbounded non-determinism. To expand on these representations, it has been necessary to study the mathematical and philosophical foundations of computation looking at claims that agent-based systems break the so called "Church-Turing Barrier" (Goldin and Wegner, 2005). It is also proposed that this type of argument from the physical to the logical is mistaken in that the causal interactions of a physical phenomenon cannot be explained at the logical level. However, it is argued that cooperation can be explained logically with the help of a CNTM. This does not trouble the Church-Turing thesis as it will always be possible to "extend" the thesis forever by squeezing refinements, as indeed Turing did himself (Turing, 1939), but without ever affecting the results of recursive function theory.

Chapter V describes the agent-based framework developed to address problem b) above. The stated aims of this thesis are to find a way to enable meta-heuristics to work together to play to each others strengths in order to produce an improving search. In so doing another aim is to build a generic modular framework and platform

that uses cooperation to solve many different problems in combinatorial optimisation. To achieve this an ontology for combinatorial optimisation was proposed and developed as well as a distributed pattern matching protocol. The ontology allows the system's heuristics and meta-heuristics to maintain a "domain barrier" (Ouelhadj and Petrovic, 2010) between the generic part of the system and the concrete problem specific details needed to solve a specific problem. This barrier is maintained so that the interface between the generic and the concrete maintained at the level of *decision variables* rather than at the level of *heuristics* as is common with hyper-heuristics.

To build the cooperation protocol it was necessary to use a novel class of parallel asynchronous algorithm where any participant can complete a given search on its own but if they cooperate with other agents then improving solutions can be found. This prosed that this class of algorithms is only possible system that exhibit this property such as agent-based systems.

The system has been tested extensively on the STSP in chapter VI, PFSP in chapter VII, and NRP in chapterVIII. It is established that cooperating agents instantiating relatively simple meta-heuristics produce better results than their stand alone counterparts. Furthermore the meta-heuristics themselves need little reconfiguring from one problem type to another. Also cooperating meta-heuristics executing a simple pattern matching protocol can use their relative strengths to produce improving solutions. It was also shown that the framework is scalable in that more agents cooperating produce better results. The system is shown to be generic, flexible and modular by being tested on a number of combinatorial optimisation problems where one was constrained.

To summarise the contributions of this thesis are:

- a new formal theory of cooperating Turing machines
- a generic flexible scalable agent-based framework for combinatorial optimisation
- a flexible, generic, pattern matching cooperation protocol using ontologies

- different cooperating meta-heuristics can by careful handling be made to work on different problems without specialised configuring
- new ontologies have been proposed
- better than state of the art results for nurse rostering for the Belgian hospital benchmarks
- impressive results for a new fairness measure.

9.1 Future work

This is an interesting and little researched topic that warrants further investigation. The work in this thesis is a first pass at building a generic agent-based platform for combinatorial optimisation. There are many more things that need to be understood, such as refining the current pattern matching protocol and finding other cooperating protocols. Here is a list of future work for the platform.

- extend the ontology and test on different problems such as vehicle routing problems or more contained problems
- establish if adding more meta-heuristics improves results
- find out if there is a pattern or trend to adding more agents to achieve better results
- improve the pattern matching protocol to share more patterns
- continue the work on fairness on agent-based systems by representing a nurse or group of nurses per agent
- continue work on finding new fairness models using objective functions and other model of nurses such as preference profiles

Finally, this framework will be published as an open source project so that other (meta-)heuristics and cooperation protocols can be added and tested by other researchers. The project is called MACS (Multi-agent Cooperative Search) and will published at the following link http://www.port.ac.uk/maths.

BIBLIOGRAPHY

G. Agha and C. Hewitt. Concurrent programming using actors. In *Object-oriented concurrent programming*, pages 37–53, 1987.

G. Agha, I. Mason, S. Smith, and C. Talcott. Towards a theory of actor computation. In *CONCUR'92*, pages 565–579, 1992.

G. A. Agha. Actors: a model of concurrent computation in distributed systems. 1985.

U. Aickelin and J. Li. An estimation of distribution algorithm for nurse scheduling. *Annals of Operations Research*, 155(1):289–309, 2007.

U. Aickelin and P. White. Building better nurse scheduling algorithms. *Annals of Operations Research*, 128:159–177, 2004.

U. Aickelin, E. K. Burke, and J. Li. An estimation of distribution algorithm with intelligent local search for rule-based nurse rostering. *Journal of the Operational Research Society*, 58(12):1574–1585, 2006.

U. Aickelin, E. K. Burke, and J. Li. An estimation of distribution algorithm with intelligent local search for rule-based nurse rostering. *Journal of the Operational Research Society*, 58(12):1574–1585, 2007.

E. Alba. *Parallel metaheuristics: a new class of algorithms*. Wiley-Interscience, 2005. ISBN 0471678066.

Y. Arafa, G. Dionisi, A. Mamdani, J. Pitt, S. Martin, and M. Witkowski. Towards building loyalty in e-commerce applications: Addressing issues on personalisation, persistence & presentation. 2000.

M. Aydin. Metaheuristic agent teams for job shop scheduling problems. *Holonic and Multi-Agent Systems for Manufacturing*, pages 185–194, 2007.

M. N. Azaiez and S. S. Al-Sharif. A 0-1 goal programming model for nurse scheduling. *Computers and Operations Research*, 32(3):507–491, 2005.

V. Bachelet and E. G. Talbi. A parallel co-evolutionary metaheuristic. *Parallel and Distributed Processing*, pages 628–635, 2000.

J. F. Bard and H. W. Purnomo. Real-time scheduling for nurses in response to demand fluctuations and personnel shortages. In E. K. Burke and M. Trick, editors, *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, PATAT, pages 67–87, Pittsburgh, August 2004.

J. F. Bard and H. W. Purnomo. A column generation-based approach to solve the preference scheduling problem for nurses with downgrading. *Socio-Economic Planning Sciences*, 39:139–213, 2005a.

J. F. Bard and H. W. Purnomo. Preference scheduling for nurses using column generation. *European Journal of Operational Research*, 164:510–534, 2005b.

J. F. Bard and H. W. Purnomo. Hospital-wide reactive scheduling of nurses with preference considerations. *IIE TRANSACTIONS*, 37:589–608, 2005c.

J. F. Bard and H. W. Purnomo. Incremental changes in the workforce to accommodate changes in demand. *Health Care Management Science*, 9:71–85, 2006.

J. F. Bard and H. W. Purnomo. Cyclic preference scheduling of nurses using a lagrangian-based heuristic. *Journal of Scheduling*, 10(1):5–23, 2007.

G. Beddoe, S. Petrovic, and J. Li. A hybrid metaheuristic case-based reasoning system for nurse rostering. *Journal of Scheduling*, 12(2):99–119, 2009.

G. R. Beddoe and S. Petrovic. Selecting and weighting features using a genetic algorithm in a case-based reasoning approach to personnel rostering. *European Journal of Operational Research*, 10(1):649–671, 2006.

G. R. Beddoe and S. Petrovic. Enhancing case-based reasoning for personnel rostering with selected tabu search concepts. *Journal of the Operational Research Society*, 58(12):1586–1598, 2007.

F. Bellanti, G. Carello, F. Della Croce, and R. Tadei. A greedy-based neighbourhood search approach to a nurse rostering problem. *European Journal of Operational Research*, 153(1):28–40, 2004.

F. Bellifemine, A. Poggi, and G. Rimassa. Jade–a fipa-compliant agent framework. 99:97–108, 1999.

F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. Jade: A software framework for developing multi-agent applications. lessons learned. *Information and Software Technology*, 50(1-2):10–21, 2008.

F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*. Wiley, 2007. ISBN 0470057475.

J. J. Bentley. Fast algorithms for geometric traveling salesman problems. *INFORMS Journal on Computing*, 4(4):387, 1992.

D. Bertsimas and J. Tsitsiklis. Simulated annealing. *Statistical Science*, pages 10–15, 1993a.

D. Bertsimas and J. Tsitsiklis. Simulated annealing. *Statistical Science*, pages 10–15, 1993b.

D. Bertsimas and J. Tsitsiklis. Simulated annealing. *Statistical Science*, 8(1):10–15, 1993c.

M. J. Bester, I. Nieuwoudt, and J. H. Van Vuuren. Finding good nurse duty schedules: a case study. *Journal of Scheduling*, 10:387–405, 2007.

P. Beullens, L. Muyldermans, D. Cattrysse, and D. Van Oudheusden. A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, 147(3):629–643, 2003.

B. Bilgin. Project web page of automation of nurse rostering in belgian hospitals, 2008. URL `http://allserv.kahosl.be/~burak/project.html`.

B. Bilgin, P. Smet, and G. Vanden Berghe. A mathematical model for a generic nurse rostering problem. Technical report, December 2011.

B. Bilgin, P. De Causmaecker, B. Rossie, and G. Vanden Berghe. Local search neighbourhoods for dealing with a novel nurse rostering model. *Annals of Operations Research*, pages 1–25, 2012.

Burak Bilgin, Peter Demeester, Mustafa Misir, Wim Vancroonenburg, and Greet Vanden Berghe. One hyperheuristic approach to two timetabling problems in health care. *Journal of Heuristics*, to appear.

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

G. Boolos, J. P. Burgess, and R. C. Jeffrey. *Computability and logic*. Cambridge Univ Pr, 2002.

S. Bourdais, Ph. Galinier, and G. Pesant. Hibiscus: A constraint programming application to staff scheduling in health care. In *CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 153–167, 2003.

A. L. Bouthillier and T. G. Crainic. A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Computers & Operations Research*, 32 (7):1685–1708, 2005.

R. Bronson and G. Naadimuthu. *Schaum's outline of theory and problems of operations research*. Schaum's Outline Series, 1997.

R. A. Brooks. Elephants don't play chess. *Robotics and autonomous systems*, 6(1): 3–15, 1990.

P. Brucker, E. K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. Adaptive construction of nurse schedules: A shift sequence based approach. *Journal of Heuristics*, 16(4):559–573, 2010.

E. K. Burke, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for the nurse rostering problem. In X. Yao et al. , editor, *Simulated Evolution and Learning 1998*, volume 1585 of *Lecture Notes in Artificial Intelligence*, pages 187–194, 1999.

E. K. Burke, P. Cowling, P. De Causmaecker, and G. Vanden Berghe. A memetic approach to the nurse rostering problem. *Applied Intelligence, Special issue on Simulated Evolution and Learning*, 15:199–214, 2001a.

E. K. Burke, P. De Causmaecker, S. Petrovic, and G. Vanden Berghe. Fitness evaluation for nurse scheduling problems. In *Proceedings of the Congress on Evolutionary Computation (CEC2001)*, pages 1139–1146, Seoul, Korea, May 27-30 2001b. IEEE Press.

E. K. Burke, P. De Causmaecker, G. V. Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of scheduling*, 7(6):441–499, 2004a. ISSN 1094-6136.

E. K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004b.

E. K. Burke, P. De Causmaecker, S. Petrovic, and G. Vanden Berghe. Metaheuristics for handling time interval coverage constraints in nurse scheduling. *Applied Artificial Intelligence*, 20(9):743–766, 2006.

E. K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A time pre-defined variable depth search for nurse rostering. Technical report, 2007.

E. K. Burke, T. Curtois, G. Post, R. Qu, and B. Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188:330–341, 2008.

E. K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A scatter search approach to the nurse rostering problem. *Journal of the Operational Research Society*, to appear.

W. Cancino, L. Jourdan, E. G. Talbi, and A. Delbem. A parallel multi-objective evolutionary algorithm for phylogenetic inference. *Learning and Intelligent Optimization*, pages 196–199, 2010.

N. Carver and V. Lesser. Evolution of blackboard control architectures. *Expert Systems with Applications*, 7(1):1–30, 1994.

P. De Causmaecker and G. Vanden Berghe. A categorisation of nurse rostering problems. *Journal of Scheduling*, 14(1):3–16, 2011.

P. De Causmaecker and G. Vanden Berghe. Towards a reference model for timetabling and rostering. *Annals of Operations Research*, to appear.

C. C. B. Cavalcante, C. Carvalho de Souza, M. W. P. Savelsbergh, Y. Wang, and L. A. Wolsey. Scheduling projects with labor constraints. *Discrete Applied Mathematics*, 112(1):27–52, 2001.

P. Y. C. Chan, M. Hiroux, and G. Weil. Strategic employee scheduling. In E. K. Burke and H. Rudová, editors, *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, PATAT, pages 157–166, 2006.

C. L. Chen and R. L. Bulfin. Complexity of single machine, multi-criteria scheduling problems. *European Journal of Operational Research*, 70(1):115–125, 1993.

M. V. Chiaramonte and L. M. Chiaramonte. An agent-based nurse rostering system under minimal staffing conditions. *International Journal of Production Economics*, 114(2):697–713, 2008.

M. Chiarandini, A. Schaerf, and F. Tiozzo. Solving employee timetabling problems with flexible workload using tabu search. In E. K. Burke and W. Erben, editors, *Proceedings of the 3th International Conference on the Practice and Theory of Automated Timetabling*, PATAT, pages 298–302, Konstanz, Germany, August 2000.

A. H. W. Chun, S. H. C. Chan, G. P. S. Lam, F. M. P. Tsang, J. Wong, and D. W. M. Yeung. Nurse rostering at the hospital authority of hong kong. In *Proceedings of the 17th National Conference on AAAI and 12th Conference on IAAI*, pages 951–956, 2000.

A. Church. A set of postulates for the foundation of logic. *Annals of mathematics*, 33(2):346–366, 1932.

S. H. Clearwater, T. Hogg, and B. A. Huberman. Cooperative problem solving. *Computation: The Micro and the Macro View*, pages 33–70, 1992.

R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of scheduling*. Dover Publications, 2003.

P. Cotogno. Hypercomputation and the physical church-turing thesis. *The British Journal for the Philosophy of Science*, 54(2):181, 2003.

P. I. Cowling, D. Ouelhadj, and S. Petrovic. A multi-agent architecture for dynamic scheduling of steel hot rolling. *Journal of Intelligent Manufacturing*, 14(5):457–470, 2003.

T. Crainic and M. Toulouse. Explicit and emergent cooperation schemes for search algorithms. *Learning and intelligent optimization*, pages 95–109, 2008.

T. Crainic, N. Hail, and Québec) Centre for Research on Transportation (Montréal. *Parallel meta-heuristics applications*. Montréal: Centre for Research on Transportation= Centre de recherche sur les transports (CRT), 2005.

T. G. Crainic and M. Gendreau. Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, 8(6):601–627, 2002.

T. G. Crainic and G. Laporte. *Fleet management and logistics*. Kluwer Academic Pub, 1998. ISBN 0792381610.

T. G. Crainic and M. Toulouse. Parallel meta-heuristics. *Handbook of metaheuristics*, pages 497–541, 2010.

T. G. Crainic, M. Toulouse, and M. Gendreau. Synchronous tabu search parallelization strategies for multicommodity location-allocation with balancing requirements. *OR Spectrum*, 17(2):113–123, 1995a.

T. G. Crainic, M. Toulouse, and M. Gendreau. Synchronous tabu search parallelization strategies for multicommodity location-allocation with balancing requirements. *OR Spectrum*, 17(2):113–123, 1995b. ISSN 0171-6468.

T. G. Crainic, M. Toulouse, and M. Gendreau. Toward a taxonomy of parallel tabu search heuristics. *INFORMS Journal on Computing*, 9:61–72, 1997.

T. G. Crainic, Y. Li, and M. Toulouse. A first multilevel cooperative algorithm for capacitated multicommodity network design. *Computers & Operations Research*, 33(9):2602–2622, 2006. ISSN 0305-0548.

T. G. Crainic, G. C. Crisan, M. Gendreau, N. Lahrichi, and W. Rei. A concurrent evolutionary approach for rich combinatorial optimization. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2017–2022, 2009.

G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

N. Cutland. *Computability, an introduction to recursive function theory*. Cambridge Univ Pr, 1980.

J. Dale and E. Mamdani. Open standards for interoperating agent-based systems. *Software Focus*, 2(1):1–8, 2001.

G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, pages 393–410, 1954.

P. De Causmaecker. Towards a reference model for timetabling and rostering. In E. K. Burke and M. Gendreau, editors, *The Practice and Theory of Automated Timetabling*, Proceedings of the 5th International Conference, Montreal, August 2008.

P. De Causmaecker and G. Vanden Berghe. Relaxation of coverage constraints in hospital personnel rostering. In E. K. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling*, volume 2740 of *Lecture Notes in Computer Science*, pages 129–147. Springer, 2003.

S. de Jong and K. Tuyls. Human-inspired computational fairness. *Autonomous Agents and Multi-Agent Systems*, pages 1–24. ISSN 1387-2532.

Y. Demazeau and J. P. Müller. *Decentralized AI: Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Cambridge, England, August 16-18, 1989*, volume 1. North Holland, 1990.

T. M. Dias, D. F. Ferber, C. C. de Sousa, and A. V. Moura. Constructing nurse schedules at large hospitals. *International Transactions in Operational Research*, 10:245–265, 2003.

M. Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2, 1999.

M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Stützle. *Ant Colony Optimization and Swarm Intelligence: 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4-7, 2006, Proceedings*, volume 4150. Springer, 2006.

A. Duenas, N. Mort, C. Reeves, and D. Petrovic. Handling preferences using genetic algorithms for the nurse scheduling problem. In G. Kendall, E. K. Burke, and S. Petrovic, editors, *Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2003)*, pages 180–196, Nottingham, UK, 2003.

R. Engelmore. *Blackboard systems: edited by Robert Engelmore, Tony Morgan.* Addison-Wesley, 1988.

Giovanni Felici and Claudio Gentile. A polyhedral approach for the staff rostering problem. *Management Science*, 50(3):381–393, 2004.

J. Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 33. Addison-Wesley Reading, MA, 1999.

R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.

T. Finin, R. Fritzson, D. McKay, and R. McEntire. Kqml as an agent communication language. page 463, 1994.

FIPA. Foundation for intelligent physical agents. *Available http://www. fipa. org/specs/fipa00023*, 2000.

FIPA. Specification, f. c. n. i. p., 2009. URL `http://www. fipa. org/specs/fipa00029`.

TCC FIPA. Fipa communicative act library specification. *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00037/SC00037J. html (30.6. 2004)*, 2008.

M. M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956.

S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. *Intelligent Agents III Agent Theories, Architectures, and Languages*, pages 21–35, 1997.

L. Gasser and M. N. Huhns. Distributed artificial intelligence, vol. ii. *London: PitmanBlaye, A., Light, P., Joiner, R. & Sheldon, S.(1991) Collaboration as a facilitator of planning and problem solving on a computer based task. British Journal of Psychology*, 9:471–483, 1989.

M. Gendreau, F. Guertin, J. Y. Potvin, and E. Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation science*, 33(4):381–390, 1999.

M. R. Genesereth, R. E. Fikes, et al. Knowledge interchange format-version 3.0: reference manual. 1992.

A. Glausch and W. Reisig. *On the Expressive Power of Unbounded Nondeterministic Abstract State Machines*. Citeseer, 2006.

A. Glausch and W. Reisig. An asm-characterization of a class of distributed algorithms. *Rigorous Methods for Software Construction and Analysis*, pages 50–64, 2009.

F. Glover. Tabu search: a tutorial. *Interfaces*, pages 74–94, 1990a.

F. Glover. Tabu search: a tutorial. *Interfaces*, pages 74–94, 1990b.

F. Glover. Tabu search: a tutorial. *Interfaces*, pages 74–94, 1990c.

F. Glover and E. Taillard. A user's guide to tabu search. *Annals of operations research*, 41(1):1–28, 1993.

D. Goldin. Persistent turing machines as a model of interactive computation. *Foundations of Information and Knowledge Systems*, pages 116–135, 2000.

D. Goldin and P. Wegner. The church-turing thesis: Breaking the myth. *New Computational Paradigms*, pages 152–168, 2005.

D. Q. Goldin, S. A. Smolka, P. C. Attie, and E. L. Sonderegger. Turing machines, transition systems, and interaction. *Information and Computation*, 194(2):101–128, 2004.

R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5(2):287–326, 1979.

M. L. De Grano, J. D. Medeiros, and D. Eitel. Accommodating individual preferences in nurse scheduling via auctions and optimization. *Health Care Management Science*, 12(3):228–242, 2009.

D. Grimshaw. Jade administration tutorial, 2011. URL `http://jade.tilab.com/doc/tutorials/JADEAdmin/index.html`.

T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

T. R. Gruber et al. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human Computer Studies*, 43(5):907–928, 1995.

Y. Gurevich. Sequential abstract-state machines capture sequential algorithms. *ACM Transactions on Computational Logic (TOCL)*, 1(1):77–111, 2000. ISSN 1529-3785.

W. J. Gutjahr and M. S. Rauner. An aco algorithm for a dynamic regional nurse-scheduling problem in austria. *Computers & Operations Research*, 34(3):642–666, 2007.

S. Haspeslagh, P. De Causmaecker, and G. Vanden Berghe. A multi-agent system handling personnel shortages in hospitals. In *Proceedings of the 4th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2009)*, MISTA, pages 693–695, Dublin, August 2009a.

S. Haspeslagh, T. Messelis, B. Bilgin, P. De Causmaecker, and G. Vanden Berghe. Hardness. Technical report, 2009b.

C. Hewitt. Viewing control structures as patterns of passing messages. *Artificial intelligence*, 8(3):323–364, 1977.

C. Hewitt. What is commitment?: physical, organizational, and social. 2006a.

C. Hewitt. The repeated demise of logic programming and why it will be reincarnated. 2006b.

C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 235–245, 1973.

D. Hilbert and W. Ackermann. Grundzüge der theoretischen logik. *Bull. Amer. Math. Soc. 59 (1953), 263-267. DOI: 10.1090/S0002-9904-1953-09701-4 PII: S*, 2(9904): 09701–4, 1953. ISSN 1088-9485.

T. Hogg and C. P. Williams. Solving the really hard problems with cooperative search. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 231–231, 1993.

J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-wesley Reading, MA, 1979.

K. INTERCHANGE. The darpa knowledge sharing effort: Progress report. *Readings in Agents*, page 243, 1998.

M. Isken. An implicit tour scheduling model with applications in healthcare. *Annals of Operations Research*, 128:91–109, 2004.

T. James, C. Rego, and F. Glover. A cooperative parallel tabu search algorithm for the quadratic assignment problem. *European Journal of Operational Research*, 195 (3):810–826, 2009.

N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1):7–38, 1998.

P. Jonsson and G. Nordh. Generalised integer programming based on logically defined relations. *Mathematical Foundations of Computer Science 2006*, pages 549–560, 2006a.

P. Jonsson and G. Nordh. Generalised integer programming based on logically defined relations. *Mathematical Foundations of Computer Science 2006*, pages 549–560, 2006b.

D. L. Kellogg and S. Walczak. Nurse scheduling: From academia to implementation or not? *Interfaces*, 37(4):355–369, 2007.

S. C. Kleene. Introduction to metamathemattes, 1952.

R. Laganiére and A. Mitiche. Parallel tabu search for robust image filtering. In *Proceedings of IEEE Workshop on Nonlinear Signal and Image Processing (NSIP'95)*, volume 2, pages 603–605, 1995.

A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

G. Laporte. A short history of the traveling salesman problem. *Canada Research Chair in Distribution Management, Centre for Research on Transportation (CRT) and GERAD HEC Montréal, Canada*, 2006.

J. Larrabee, J. Sions, M. Fanning, M. Withrow, and A. Ferretti. Evaluation of a program to increase evidence-based practice change. *Journal of Nursing Administration*, 37(6):302–310, 2007.

P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13:129–170, 1999.

S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973a.

S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, pages 498–516, 1973b.

J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations research*, pages 972–989, 1963.

C. C. Lo, C. C. Lin, T. J. Dai, and D. Wong. Artificial immune systems for intelligent nurse rostering. In *Proceedings*, IEEE International Conference on Industrial Engineering and Engineering Management, pages 862–866, Singapore, December 2009.

E. J. Lodree Jr., C. D. Geiger, and X. Jiang. Taxonomy for integrating scheduling theory and human factors: Review and research opportunities. *International Journal of Industrial Ergonomics*, 39(1):39–51, 2009.

B. Maenhout and M. Vanhoucke. Comparison and hybridization of crossover operators for the nurse scheduling problem. *Annals of Operations Research*, 159(1):333–353, 2008.

M. Malek, M. Guruswamy, M. Pandya, and H. Owens. Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research*, 21(1):59–84, 1989.

R. Malek. An agent-based hyper-heuristic approach to combinatorial optimization problems. In *Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on*, volume 3, pages 428–434, 2010.

Simon Martin, Djamila Ouelhadj, Patrick Beullens, and Ender Ozcan. A generic agent-based framework for cooperative search using pattern matching and reinforcement learning. Technical report, University of Portsmouth, January 2012.

D. Meignan, A. Koukam, and J. C. Créput. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, pages 1–21. ISSN 1381-1231.

D. Meignan, J. C. Creput, and A. Koukam. A coalition-based metaheuristic for the vehicle routing problem. pages 1176–1182, 2008.

D. Meignan, A. Koukam, and J. C. Créput. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, pages 1–21, 2010. ISSN 1381-1231.

T. Messelis, S. Haspeslagh, B. Bilgin, P. De Causmaecker, and G. Vanden Berghe. Towards prediction of algorithm performance in real world optimisation problems. In *Proceedings of the 21st Benelux Conference on Artificial Intelligence*, number 21 in BNAIC, pages 177–183, Eindhoven, October 2009.

H. Meyer auf'm Hofe. Nurse rostering as constraint satisfaction with fuzzy constraints and inferred control strategies. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 57:67–99, 2001a.

H. Meyer auf'm Hofe. Solving rostering tasks as constraint optimization. In E. K. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling, Third International Conference*, volume 2079 of *Lecture Notes in Computer Science*, pages 191–212, 2001b.

M. Milano and A. Roli. Magma: A multiagent architecture for metaheuristics. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(2): 925–941, 2004. ISSN 1083-4419.

R. Milner. *A calculus of communicating systems*. Springer-Verlag New York, Inc., 1982.

D. C. Montgomery. *Design and analysis of experiments*. John Wiley & Sons, 2008.

M. Moz and M. V. Pato. Solving the problem of rerostering nurse schedules with hard constraints: New multicommodity flow models. *Annals of Operations Research*, 128: 179–197, 2004.

M. Moz and M. V. Pato. A genetic algorithm approach to a nurse rerostering problem. *Computers & Operations Research*, 34(3):667–691, 2007.

C. W. Mueller and J. C. McCloskey. Nurses job satisfaction, a proposed measure. *Nursing Research*, 39:113–117, 1990.

M. Nawaz, E. Enscore, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983a.

M. Nawaz, E. E. Enscore Jr, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983b.

H. S. Nwana, D. T. Ndumu, et al. A perspective on software agents research. *The Knowledge Engineering Review*, 14(2):125–142, 1999.

T. Ord. The many forms of hypercomputation. *Applied mathematics and computation*, 178(1):143–153, 2006.

T. Osogami and H. Imai. Classification of various neighborhood operations for the nurse scheduling problem. Technical Report 135, 2000.

D. Ouelhadj and S. Petrovic. A cooperative distributed hyper-heuristic framework for scheduling. In *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*, pages 2560–2565, 2008.

D. Ouelhadj and S. Petrovic. A cooperative hyper-heuristic search framework. *Journal of Heuristics*, 16(6):835–857, 2010.

D. Ouelhadj, S. Martin, P. Smet, E. Ozcan, and G. Vanden Berghe. Fairness in nurse rostering. 2012.

E. Ozcan. Memetic algorithms for nurse rostering. In *Computer and Information Sciences - Proceedings ISCIS*, number 3733 in Lecture Notes in Computer Science, pages 482–492, 2005.

E. Özcan and M. Erentürk. A brief review of memetic algorithms for solving euclidean 2d traveling salesrep problem. In *The 13th Turkish Symposium on Artificial Intelligence and Neural Networks*, pages 99–108, 2004.

C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Dover Pubns, 1998.

D. Parr and J. M. Thompson. Solving the multi-objective nurse scheduling problem with a weighted cost function. *Annals of Operations Research, Special Issue on Personnel Scheduling and Planning*, 155(1):279–288, 2007.

S. Petrovic and G. V. Berghe. Comparison of algorithms for nurse rostering problems. In *Procedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, pages 1–18, 2008.

S. Petrovic and G. Vanden Berghe. A comparison of two approaches to nurse rostering. *Annals of Operations Research*, 2012.

M. Pinedo. *Scheduling: theory, algorithms, and systems.* Prentice-Hall, New Jersey, 2002.

j. Pitt, Y. ARAFA, G. DIONISI, S. MARTIN, and M. WITKOWSKI. Creating loyalty in e-commerce using agent technology.

S. Poslad, P. Buckle, and R. Hadingham. The fipa-os agent platform: Open source for open standards. In *Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, volume 355, page 368, 2000.

J. Puchinger and G. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. *Artificial intelligence and knowledge engineering applications: a bioinspired approach*, pages 113–124, 2005.

P. Punnakitikashem, J. M. Rosenberger, and D. B. Behan. Stochastic programming for nurse assignment. *Computational Optimization and Applications*, 40:321–349, 2008.

Abdur Rais and Ana Viana. Operations research in healthcare: a survey. *International Transactions in Operational Research*, 18(1):1–31, 2011.

G. Reinelt. Tsplib–a traveling salesman problem library. *INFORMS Journal on Computing*, 3(4):376, 1991.

G. Reinelt. *The traveling salesman: computational solutions for TSP applications.* Springer-Verlag, 1994.

E. A. Robb, A. C. Determan, L. R. Lampat, M. J. Scherbring, R. M. Slifka, and N. A. Smith. Self-scheduling: Satisfaction guaranteed. *Nursing Management*, 34 (7):16–18, 2003.

L. M. Rocha. Artificial semantically closed objects. *Communication and Cognition-AI*, 12(1-2):63–90, 1995.

Elina Rönnberg and Torbjörn Larsson. Automating the self-scheduling process of nurses in swedish healthcare: a pilot study. *Health Care Management Science*, 13: 35–53, 2010.

R. Ruiz and T. Stutzle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

Rubén Ruiz and Concepción Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165 (2):479–494, 2005.

B. Russell and A. N. Whitehead. *Principia mathematica*. Cambridge University Press Cambridge, UK, 1968.

J. R. Searle. *Speech acts: An essay in the philosophy of language*. Cambridge university press, 1970. ISBN 052109626X.

J. R. Searle. *Expression and meaning: Studies in the theory of speech acts*. Cambridge Univ Pr, 1985.

O. Shagrir. Godel on turing on computability. *Church's Thesis after 70 years*, page 393, 2006.

U. Shardanand and P. Maes. Social information filtering: algorithms for automating "word of mouth". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217, 1995.

S. P. Siferd and W. C. Benton. Workforce staffing and scheduling: Hospital nursing specific models. *European Journal of Operational Research*, 60:233–246, 1992.

R. Silvestro and C. Silvestro. An evaluation of nurse rostering practices in the national health service. *Journal of Advanced Nursing*, 32(3):525–525, 2000.

R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on*, 100(12):1104–1113, 1980.

F. C. N. I. P. Specification. At: http://www. fipa. org/specs/fipa00029. *SC00029H. pdf*.

F. C. N. I. P. Specification. Link: http://www. fipa. org/specs/fipa00029. *SC00029H. html*, 2003.

M. Stannett. Computation and hypercomputation. *Minds and Machines*, 13(1):115–153, 2003.

K. Steiglitz and P. Weiner. Some improved algorithms for computer solution of the traveling salesman problem. 1968.

T. Stützle and H. Hoos. Max-min ant system and local search for combinatorial optimization problems. 1997.

K. Sycara, A. Pannu, M. Willamson, D. Zeng, and K. Decker. Distributed intelligent agents. *IEEE expert*, 11(6):36–46, 1996.

A. Syropoulos. *Hypercomputation: computing beyond the Church-Turing barrier*. Springer-Verlag New York Inc, 2008.

E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, 1990a. ISSN 0377-2217.

E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, 1990b.

E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

E. D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA journal on Computing*, 6(2):108–117, 1994.

E. G. Talbi and V. Bachelet. Cosearch: A parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms*, 5(1):5–22, 2006. ISSN 1570-1166.

S. Talukdar, L. Baerentzen, A. Gove, and P. De Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4(4):295–321, 1998.

S. Talukdar, S. Murthy, and R. Akkiraju. Asynchronous teams. *Handbook of Metaheuristics*, pages 537–556, 2003.

V. T'kindt, V. Tk*indt, and J. C. Billaut. *Multicriteria scheduling: theory, models and algorithms*. Springer Verlag, 2006.

S. Topaloglu and H. Selim. Nurse scheduling using fuzzy multiple objective programming. In H. G. Okuno and M. Ali, editors, *IEA/AIE 2007*, volume 4570 of *Lecture Notes in Artificial Intelligence*, pages 54–63. Springer-Verlag, 2007.

M. Toulouse and T. G. Crainic. B. sans o. an experimental study of systemic behavior of cooperative search algorithms. *Meta-Heuristics: Theory and Applications*, pages 373–392.

M. Toulouse, K. Thulasiraman, and F. Glover. Multi-level cooperative search: A new paradigm for combinatorial optimization and an application to graph partitioning. *Euro-Par'99 Parallel Processing*, pages 533–542, 1999.

A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(1):230, 1937.

A. M. Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, 2(1):161–228, 1939.

A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

A. M. Turing. Lecture to the london mathematical society on 20 february 1947. *MD COMPUTING*, 12:390–390, 1995.

E. Vallada and R. Ruiz. Cooperative metaheuristics for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 193(2):365–376, 2009.

E. Vallada and R. Ruiz. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega*, 38(1):57–67, 2010.

J. Van Leeuwen and J. Wiedermann. The turing machine paradigm in contemporary computing. *Mathematics Unlimited-2001 and Beyond. Springer-Verlag, Berlin*, pages 1139–1155, 2001.

M. Vanhoucke and B. Maenhout. - a nurse scheduling problem library: a tool to evaluate ($meta-$)heuristic procedures. In S. Brailsford and P. Harper, editors, *Operational Research for Health Policy: Making Better Decisions*, Proceedings of the 31st Annual Meeting of the working group on Operations Research Applied to Health Services, pages 151–165, 2007.

M. Vanhoucke and B. Maenhout. On the characterisation and generation of nurse scheduling problem instances. *European Journal of Operational Research*, 196(2): 457–467, 2009.

J. Antonio Vázquez-Rodríguez and Gabriela Ochoa. On the automatic discovery of variants of the neh procedure for flow shop scheduling using genetic programming. *Journal of the Operational Research Society*, 62:381–396, 2011.

V. y. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985.

MichałWalicki and Sigurd Meldal. Algebraic approaches to nondeterminism–-an overview. *ACM Comput. Surv.*, 29(1):30–81, March 1997. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/248621.248623. URL `http://doi.acm.org/10.1145/248621.248623`.

Z. G. Wang and C. Wang. Automating nurse self-rostering: A multiagent systems model. In *2009 IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS*, volume 1–9 of *SMC 2009*, pages 4422–4425, 2009.

M. Warner. Nurse staffing, scheduling, and reallocation in the hospital. *Hospital & Health Services Administration*, pages 77–90, 1976.

P. Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, 1997.

P. Wegner. Interactive foundations of computing. *Theoretical computer science*, 192 (2):315–351, 1998.

P. Wegner and D. Goldin. Computation beyond turing machines. *Communications of the ACM*, 46(4):100–102, 2003.

G. Y. C. Wong and A. H. W. Chun. Constraint-based rostering using meta-level reasoning and probability-based ordering. *Engineering Applications of Artificial Intelligence*, 17(6):599–610, 2004.

M. Woodridge and N. R. Jennings. Intelligent agents: Theory and practice the knowledge engineering review. 1995.

M. Wooldridge. *An introduction to multiagent systems*. Wiley, 2009.

M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(02):115–152, 1995.

X. F. Xie and J. Liu. Multiagent optimization system for solving the traveling salesman problem (tsp). *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(2):489–502, 2009.

Su Chuan Yuan, Ming Chih Chou, Chiou Jong Chen, Yen Ju Lin, Mei-Chu Chen, Hung-Hsin Liu, and Hsien-Wen Kuo. Influences of shift work on fatigue among nurses. *Journal of Nursing Management*, 19(3, SI):339–345, 2011.

APPENDIX
ABBREVIATIONS

TSP Travelling Salesman Problem
STSP Symmetrical Travelling Salesman Problem
ATSP Asymmetrical Travelling Salesman Problem
PFSP Permutation Flow-shop Scheduling
NR Nurse Rostering
NRP Nurse Rostering Problem
TS Tabu Search
SA Simulated Annealing
VNS Variable Neighbourhood Search
TM Turing Machine
DTM Deterministic Turing Machine
NTM Non-Deterministic Turing Machine
CNTM Cooperating Non-Deterministic Turing Machine
CA Cooperating Algorithm
CMAX Maximum Makespan
NEH Nawaz, M. and Enscore Jr, E. E. and Ham, I. Designers of an Algorithm known as NEH (Nawaz et al., 1983b)
AMS Agent Management System
DF Directory Facilitator
MAOS An agents-based system for solving the TSP developed by Xie et al 2009. (Xie and Liu, 2009)
GB Gigabyte
RAM Random Access Memory
WO Weighted Sum Objective function
FO Fair Objective Function
IO Individual Weighted Sum Objective function
WIO Individual Weighted Sum Objective function per agent
FIPA Foundation for Intelligent Physical Agents
JADE JAVA Agent Development Framework
KQML Knowledge Query Manipulation Language
KIF Knowledge Interchange Format
DARPA Defence Advanced Research Projects Agency
FIPA-CL FIPA Agent Communication Language
DAI Distributed Artificial Intelligence