A generic agent-based framework for cooperative search using pattern matching and reinforcement learning

Simon Martin[a,*], Djamila Ouelhadj[a], Patrick Beullens[b], Ender Özcan[c]

[a] *Logistics and Management Mathematics Group, Department of Mathematics, University of Portsmouth, Lion Gate Building, Lion Terrace, Portsmouth, PO1 3HE.*
[b] *School of Mathematics - School of Management, University of Southampton, SO17 1BJ, United Kingdom*
[c] *Automated Scheduling, Optimisation and Planning (ASAP), School of Computer Science, University of Nottingham, Jubilee Campus, Wollaton Rd Nottingham, NG8 1BB.*

Abstract

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 study, we propose 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. 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 has been so far tested on Permutation Flow-shop Scheduling and Travelling Salesman Problem instances yielding promising results.

Keywords: combinatorial optimization, multi-agent systems, scheduling, (meta-)heuristics, cooperative search, reinforcement learning.

1. Introduction

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

*Simon Martin
Email addresses: Simon.Martin@port.ac.uk (Simon Martin), Djamila.Ouelhadj@port.ac.uk (Djamila Ouelhadj), P.Beullens@soton.ac.uk (Patrick Beullens), Ender.Ozcan@nottingham.ac.uk (Ender Özcan)

either parameter tuning and/or design of new neighbourhood operators for the new problem domain. There is almost no guidance available to choose 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. On the other hand, using a framework enabling the use of different (meta-)heuristics could result in an improved search methodology and increase 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).

The 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).

This study investigates into an agent-based (multi-agent) distributed search methodology performing cooperative search. The main goal of the study is to design a framework achieving *generality*, *flexibility* and *scalability*. The framework should be general enough to solve a variety of combinatorial optimisation problems, flexible enough to enable adding/ removing (meta-)heuristics as agents easily, and scalable enough to obtain better quality solutions as the number of (meta-)heuristic agents increases within the system. A framework carrying these characteristics is implemented using open standards. Additionally, different (meta-)heuristic agents are implemented for two problem domains as case studies: the symmetrical travelling salesman problem (STSP) and permutation flow-shop scheduling (PFSP). The computational experiments are performed on the standard benchmark problems from each domain to test the framework showing that it is in fact general, flexible and scalable, and to observe the influence of cooperation, rather than outperforming the state-of-the-art methodologies which are particularly tailored for these problem domains.

Cooperative search, its difference from the other parallel processing techniques and our motivation to develop an agent-based framework are introduced in section 2. Section 3 describes the proposed multi-agent framework for cooperative search. Section 4.1 discusses the specifics of ontologies and communication protocol used within the proposed framework. Section 5 describes how the system is implemented. Section 6 provides an overview of the STSP and PFSP problem domains. In section 7, the results of our computational experiments for solving STSP and PFSP as case studies are presented. Finally, section 8 presents conclusions and suggestions for future work.

2. Cooperative (meta-)heuristic Search

2.1. Literature Overview

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 and simulated annealing (Crainic and Laporte, 1998; Crainic and Toulouse, 2010; Alba, 2005; 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.

Crainic and Toulouse (2010) proposed three basic strategies for implementing parallel meta-heuristics:

- Type 1 - *low level parallelism*, where a meta-heuristic is implemented on a master process which distributes difficult or time consuming computations to slave processes. At each iteration, the slave processes evaluate new moves in a neighbourhood for the master to generate a new best-so-far solution;
- Type 2 - *domain decomposition*, where the search space is partitioned into different instances of a problem that are processed by slave heuristics. Each of these will communicate solutions to a master process. The master then determines the best solution;
- Type 3 - *multi-threaded strategies*, where different threads run multiple versions of the same meta-heuristics that may start from the same or different places within the search space. The threads may communicate during or at the end of execution.

On the face of it, the type of agent-based approach described in this paper is just an example of a type 3 parallel system as defined in the taxonomy above. However, 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 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. For this reason agent systems are often, but not always, distributed over many machines or even the internet.

This means that an agent is not some kind of homunculus performing a task or subtask for some other controlling process. However in the literature, the term *agent*, is often used, as above, to describe distributed sub-processes. In this paper a 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.

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 described 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.

The main challenge for cooperative search is designing cooperative mechanisms that enable the useful exchange of information between agents. 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 such as multi-commodity location with balancing requirements (Crainic et al., 1995), capacitated network design (Crainic et al., 2006) and quadratic assignments (James et al., 2009).

Comparative literature studies conducted by Crainic et al. (1997) have compared distributed synchronous and asynchronous cooperative search with sequential search. They have found the solutions obtained to be superior. Furthermore they concluded that, of the methods they discussed, asynchronous cooperative search was the most effective.

From the literature it is clear that cooperation leads to better solutions. It is also robust, covering more of the search space and often leads to better performance, or speed-up (Alba, 2005).

2.2. Motivation

In the literature so far most cooperation mechanisms are described as taking place through some pool or adaptive memory which is different to a message passing multi-agent system (Talbi and Bachelet, 2006; Milano and Roli, 2004; Meignan et al., 2008). It can also take the form of swapping whole solutions or possibly the rating of different moves (Meignan et al., 2010). To the best of our knowledge 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) 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. However, all three systems are designed for specific problem domains and use the best known (meta-)heuristics for these domains.

Malek (2010) proposed a multi-agent framework able to work on different problem domains, but the system seems to pass whole solutions through a solution pool. It is not clear whether this is type 3 system in the taxonomy of Crainic and Toulouse (2010) or a full message passing agent system.

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 to agents cooperating directly and asynchronously where the communication is used to tune parameters of algorithms and the adaptive selection of moves with parameters.

In this paper, the authors propose a generic framework of cooperating agents using pattern matching and reinforcement learning. As such 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 by asynchronous message passing to converge towards good quality solutions. The focus of this research is to develop a framework for automatic heuristic selection and parameter setting to support cooperating (meta-)heuristic agents, but not on speed-up. A further contribution of this research is to show that direct asynchronous cooperation using autonomous multi-agents can be used to great affect achieving acceptable results in different problem domains; namely permutation flow-shop scheduling and the symmetric travelling salesman problem.

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.

3. A generic multi-agent framework for cooperative search

The proposed generic multi-agent framework for cooperative search makes use of two types of agent: *launcher* and *(meta-)heuristic* agents (Figure 1).

- The launcher agent is responsible for queueing the problem instances to be solved for a given domain, configuring the (meta-)heuristic agents, passing a given problem instance to the (meta-)heuristic agents and gathering the solutions from the (meta-)heuristic agents for a given problem instance. It reads the problem data files as input for a given problem domain and sends each instance successively to the (meta-)heuristic agents to improve upon.

- 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 combinations of heuristics and meta-heuristics and their parameter settings are all defined in a configuration file the agent reads on launching. In this way, while conducting a search, the agents can each execute different meta-heuristic and heuristic combinations with different parameter settings.

A search becomes a series of messages passed between the agents. Each message is sent as a consequence of internal processing conducted by an agent. This sequence of processing and message passing conducted by the agents amounts to what is in effect a distributed algorithm. This process will be called a *conversation* throughout this paper. 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 to an output file. The launcher then moves on to the next problem.

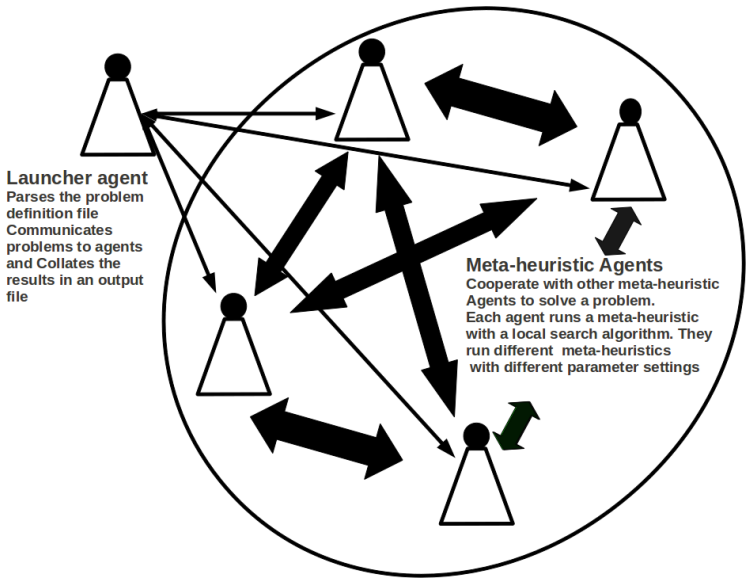


Figure 1: The generic multi-agent framework

Looking at Figure 2, irrespective of whether they are a launcher or (meta-)heuristic agent,it can be seen that an agent works by reading in a configuration file when it launches. It then executes a conversation which will continue until the agent has performed a given number of conversations. The number of conversations is provided in the launcher agent's configuration file and communicated to the (meta-)heuristic agents when initially it sends the problem to the (meta-)heuristic agents.

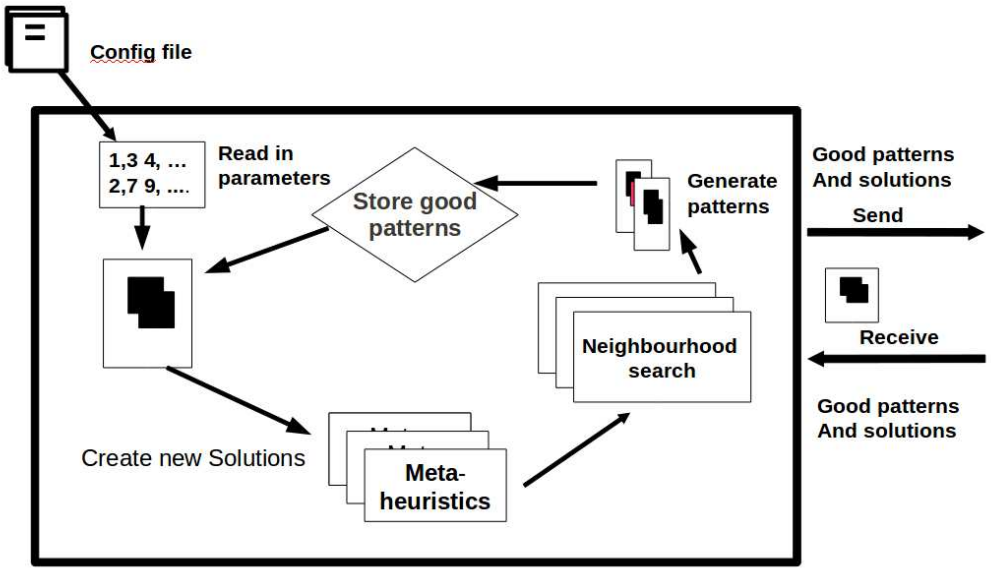


Figure 2: A (meta-)heuristic agent from the framework

4. Cooperation by pattern matching and reinforcement learning

As the framework is designed to work on many different problem types, a novel cooperation strategy has been proposed that does not just include the passing of whole solutions between agents, as this has limited success. This strategy has been devised where agents cooperate by looking for patterns that are the constituent parts of a *good* solution. These are shared amongst the agents which then build new solutions based on these patterns.

Most of the combinatorial optimisation problems basically consist of permutations, or solutions, these terms will be used interchangeably, that must be re-arranged according to constraints that define the problem. Good solutions are those permutations that give a better value with respect to an objective function on than that of a previous one. New permutations can be generated by perturbation or greedy heuristics.

For example, in the PFSP the jobs form such a permutation, while with STSP, it is the list of cities themselves. Given such a permutation, it is always possible to generate n pairs from it. For example, take the permutation where $n = 10$: <2,4,7,6,5,8,9,0,1,3>, the following n pairs can be generated:

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

.

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. The agents can then each compare pairs generated from their own solutions with pairs generated by the other agents.

The idea behind the cooperation protocol is that given an optimising search scenario, each agent takes its current best permutation or solution and generates n pairs from it. All the pairs from each agent are pooled and scored based on how frequently they appear in the pool. Only those pairs that have the highest frequency score are shared out amongst the agents. The agents then use these pairs to generate a new solution using a greedy heuristic. This process is achieved through a *conversation*.

In this way good patterns are retained from one conversation to another. This means that as the search progresses more patterns are retained and the agents converge to good solutions. It can be seen that this pattern matching algorithm makes use of a distributed form of reinforcement learning.

4.1. Ontologies

Before proceeding to the description of a conservation, it is important to discuss the role ontologies play within the framework. Ontologies are important because they raise the generality of the conversations. They are not domain specific, but define a set of general representational primitives to model some domain (Gruber et al., 1993) and as such are semantic. The idea is that because agents are autonomous and operate asynchronously, they have to cooperate with other agents on other platforms in different locations. By using ontologies they can communicate content and the means to act upon it all within the same message. This makes agents very flexible: an important property for a framework that aims to solve many problems in combinatorial optimisation.

The ontology currently used by the framework generalises the notions of: *permutations*, *pairs*, and *individual elements*. In the ontology these are called *SolutionData, HeuristicData* and *NodeData* objects respectively.

NodeData object: A NodeData object is the basic object in the ontology. It represents the individual elements of a permutation. For example, in the PFSP domain, a NodeData object represents a job, while in TSP domain it represents a city.

HeuristicData: A HeuristicData object is more complex in that it contains two NodeData objects. It represents a pair of elements in a permutation and is fundamental in the pattern matching algorithm. Also it stores other information about the pair. These include the distance between the two nodes and also the frequency score.

SolutionData: A SolutionData represents a permutation that is being optimised. It holds the current value of permutation with respect to the objective function being used. SolutionData objects also contain lists of HeuristicData and NodeData objects.

These are the only objects used in a conversation. In this way the framework is very generic and flexible because the system uses the same objects no matter what problem type it is trying to solve.

4.2. Conversation

Figure 3 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.

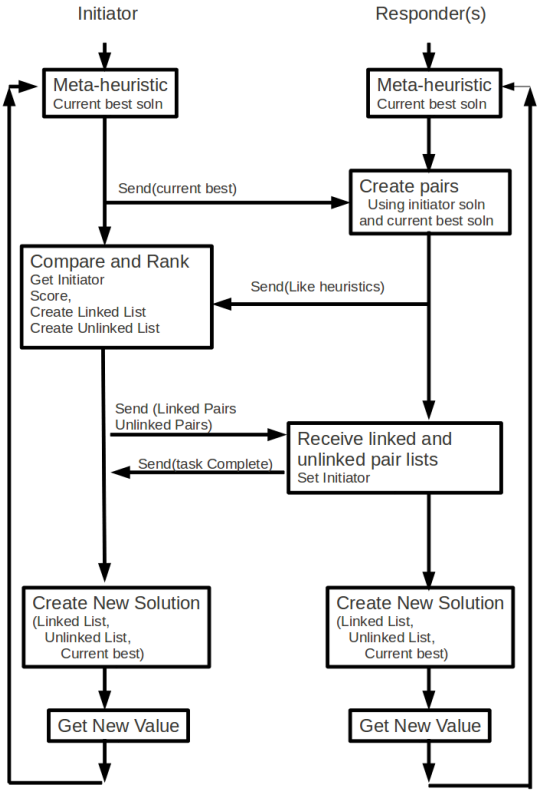


Figure 3: The Cooperation Protocol

An agent taking on the role of initiator starts a conversation. It takes a new permutation either generated from a previous conversation or supplied by the launcher agent. The new permutation or solution is then improved by the chosen (meta-)heuristic for that agent. When an improved solution is generated, it is sent to the other (meta-)heuristic agents.

The other (meta-)heuristic agents have also generated their best-so-far permutations using their designated (meta-)heuristics. They each receive the solution from the initiator and break it up into edges or pairs, as explained above in section 4, and do the same to their own best solution. The pairs are then compared and only those that are the common to both permutations are kept.

HeuristicData objects are created from these pairs storing the first and second elements of the pair. These are then sent by the receiving (meta-)heuristic agents 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.

The initiator receives the HeuristicData objects from the responding agents and pools them. Each HeuristicData object is scored by counting how frequently it occurs in the pool. Tests show that it is best to only accept HeuristicData objects with a frequency score of one less than the number of (meta-)heuristic agents (algorithm 1).

```
Input: List of HeuristicData Objects
Output: Set of HeuristicData Objects
foreach HeuristicData object in List do
    score = frequency 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

The initiator then tries to build a linked list from these high scoring HeuristicData objects. 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). Any HeuristicData objects not linked in this way are stored in an unlinked list (5,9) (3,8).

The initiator also determines which (meta-)heuristic agent is going to be the initiator in the next conversation. This is done by pooling all the values the best-so-far solutions sent by each (meta-)heuristic agent and then identifying which (meta-)heuristic agent has the best objective function value. The (meta-)heuristic agent with the best value will be the new initiator in the next conversation. The initiator then sends these lists of linked and unlinked 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 list of HeuristicData objects. Both initator and receiver (meta-)heuristic agents then create a new solution using both the linked list and the unlinked list, as well as their current best solution (algorithm 2). 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 this way a new unique permutation is generated and the objective function value is calculated.

```
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

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.

5. Implementation of the agent-based distributed framework for cooperative search

The agent-based distributed framework for cooperative search is implemented on the open source FIPA compliant development platform JADE (Bellifemine et al., 2007). The environment also comes with two important predefined agents. The *Agent Management System* (AMS); it is the main authority for the platform and is the only agent that can create and kill agents and shut down the platform. The *Directory Facilitator* (DF) advertises the services of agents on the platform so that other agents can find them. The platform also comes with 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. A JADE agent is a programme that schedules behaviours. A behaviour is a task that an agent executes. It is possible to schedule when and in what order these behaviours are executed. A behaviour will often involve an agent communicating with other agents. To this end JADE has

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 Specification. 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*.

The system 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. All problem data are parsed by the launcher agent into SolutionData HeuristicData and NodeData objects. Then, through the use of the ontology, these objects are translated seamlessly to XML which JADE uses as its messaging protocol. The (meta-)heuristic agents then de-parse these messages back into SolutionData, HeuristicData and NodeData objects. These are then kept in memory in a singleton object which is done for efficiency purposes. The (meta-)heuristic agents then just manipulate pointers to the objects held in the singleton for data information. For example, if the system wants to calculate the objective function, the data necessary is held in the singleton. As far as the system is concerned, it only deals with these objects held in memory or pointers to them. This means that to run a different problem type there is no problem specific reprogramming required. All that needs to be changed are the choices made in the configuration files for the launcher and (meta-)heuristic agents.

Logging facilities can be specified in the configuration files. The system is able to print out the progress of the search to the screen or write it to 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 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:

- Meta-heuristic: tabu search (Glover, 1990) or simulated annealing (Bertsimas and Tsitsiklis, 1993). These are very basic implementations of these algorithms.
- Number of iterations the meta-heuristic will perform
- Size of the tabu list.
- Temperature function for simulated annealing: geometric or logarithmic.
- Local search heuristic: 2-opt or hill climber with the following moves: swap, insert, reverse and shift.
- Screen debug information and search progress printed to file.

6. Case Studies

6.1. Travelling salesman problem

Given an n *by* n symmetric matrix of distances between n cities, the task for the symmetric travelling salesman problem (TSP) 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 $(\sigma_i, \sigma_{i+1}) \in E$ 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, therefore is just:

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

6.2. Permutation flow-shop scheduling

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).

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

$$C_{1,1} = p_{1,1} \tag{2}$$

$$C_{1,i} = C_{1,i-1} + p_{1,i}, \text{ where } i = 2, ..., m \tag{3}$$

$$C_{j,i} = max(C_{j,i-1}, C_{j-1,i}) + p_{j,i}, \text{ where } i = 2, ..., m, \text{ and } j = 2, ..., n \tag{4}$$

$$C_{max} = C_{n,m} \tag{5}$$

There are $n!$ possible sequences and therefore the problem is known to be NP-complete (**?**). 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., 1983; Ruiz and Maroto, 2005; Vallada and Ruiz, 2009; Vázquez-Rodríguez and Ochoa, 2011).

7. Computational experiments

7.1. Experimental settings

To evaluate the performance of the framework, we conducted several experiments using the symmetric Travelling Salesman Problem (STSP) and the Permutation Flow-shop Scheduling Problem (PFSP). Each case study was tested using well known benchmark problems.

The objectives of the experiments conducted were as follows:

1. Test if the system is flexible enough to work on different types of problem domains;
2. Test if cooperation produced better results than no cooperation;
3. Test if increasing the number of agents cooperating produced better result

Vallada and Ruiz (2009) argued that it is impossible to compare their cooperative algorithms with comparable parallel algorithms because they are implemented on parallel processing machines purely for the benefits of speed-up. For the same reasons the authors in this paper adopt a similar testing regime for testing the cooperative pattern matching used by the framework.

To facilitate testing the cooperation protocol some simple (meta-)heuristics have been developed. 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.

For the experiments, we have considered the following scenarios:

- 1 agent in stand alone mode;
- 5 agents, 1 launcher and 4 (meta-)heuristic agents;
- 9 agents, 1 launcher and 8 (meta-)heuristic agents;
- 13 agents, 1 launcher and 12 (meta-)heuristic agents;

To evaluate the performance of the proposed framework, the average percentage increase of the objective function value of the solution ($Method_{sol}$)produced by the cooperative search was calculated and compared with the known upper bound or optimum ($Best_{sol}$). An upper bound is the best known solution to a problem, however it has not been proved to be the optimum (equation 6). With the PFSP the objective was the minimisation of the total makespan, while with STSP it was the minimisation of the distances between the all the cities to produce the shortest possible tour of all the cities.

$$\frac{Method_{sol} - Best_{sol}}{Best_{sol}} \tag{6}$$

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.

The tests were conducted first, with (meta-)heuristic agents cooperating and then with (meta-)heuristic agents in stand alone mode. Each problem instance was tested five times, first with cooperation and then without cooperation. In cooperation mode half of the (meta-)heuristic agents ran simulated annealing with a geometric temperature function, and the other half ran tabu search with a tabu tenure of 7. In each case, the meta-heuristic conducted a maximum of 500 iterations. There were no differences in each of the cooperation scenarios of 12, 8 and 4 (meta-)heuristic agents cooperating.

The authors found thorough testing that for PFSP agents needed 3000 conversations. It was found that agents needed this many conversations to converge to a good solution. As well as this, extra starting and conversations were needed where the (meta-)heuristic receives and send data from and to the launcher. This is because a conversation is made up of three FIPA standard protocols.

A starting conversation is where a (meta-)heuristic agent receives data from the launcher and uses an ending conversation is where the (meta-)heuristic agent sends data back to the launcher. In both cases a FIPA Request protocol was used. The main body of the conversation was conducted 3000 times using the FIPA Contract Net Protocol. In total therefore, the (meta-)heuristic agents conducted 3002 conversations.

In the case of STSP it was found through testing that 5002 conversations were required including the usual starting and ending conversations. All conversations work the same way so the same FIPA protocols were use as those described above. However more conversations were needed for the system to converge to a solution.

In standalone mode, the (meta-)heuristic agents ran for as many iterations as all the (meta-)heuristic agents collectively in cooperation mode. This means that if a system of 12 (meta-)heuristic agents had 3002 conversations where each (meta-)heuristic agent ran its (meta-)heuristic for 500 iteration, then the equivalent standalone (meta-)heuristic agent would evaluate its (meta-)heuristic $3002 \times 12 \times 500 = 18012000$ times. The standalone (meta-)heuristic agents always executed the same number of iterations as 12 (meta-)heuristic agents cooperating scenario. This was done even when they were being compared with the 8 or 4 (meta-)heuristic agents scenarios. This gave the standalone (meta-)heuristic agents the best chance to produce a good result as it produced the most iterations for the stand alone scenario to compare with the cooperating scenarios. Also the stand alone (meta-)heuristic agents were run against each problem instance twice; once, with the (meta-)heuristic agent running simulated annealing and once with tabu search. In this way, a fair comparison could be made between cooperation and no cooperation.

7.2. The travelling salesman problem

Reinelt (1991) has produced a set of benchmark problems for all the different types travelling salesman problem including a set of problems for the STSP.

There are at least 50 problems in just the STSP library alone and there are different types of problems with the STSP set according to the coordinates provided for the position of a city. These coordinates can be given in 2 dimensions or 3 dimensions. Furthermore, the distance between cites can be calculated in different ways, such as, the euclidean distance, the manhattan distance, the maximum distance, the geographical and the pseudo-euclidean distance.

The authors chose 5 STSP problems, all using the euclidean distance for 2 dimensions. The instances chosen are: eil101 (101 cities), a280 (280 cities) gil262(262 cities), pr209 (209 cities)and rat575 (575 cites).

The local search algorithm used for these tests was a version of the 2-opt algorithm proposed by Beullens et al. (2003) with active marking and neighbour lists. A 2-opt algorithm was chosen for STSP because it is a well known local search heuristic for this problem.

Each problem was run 5 times for groups of 12,8 and 4 cooperating agents and 1 non cooperating agent. As before each problem result was averaged and the average percentage increase from the known optimum was calculated. Therefore, Table 1 shows the average percentage increase above the known optima for these problems.

It can be seen clearly that there is a 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 taillored to solve these benchmark problems to optimality, it can still with relatively simple (meta-)heuristics significantly improve the results.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. .

Table 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
eil101	3.6884	2.7027	4.1335	10.0159	10.0159
a280	13.5712	14.1683	13.0593	15.8201	15.8201
gil262	14.8276	14.8528	15.2902	15.4331	15.4331
pr299	15.7976	16.2873	17.3252	18.0615	18.0615
rat575	18.1810	18.4393	18.6476	18.8100	18.8100

Figure 4 is a graph of boxplots showing the analysis of spread of variance from the optima using ANOVA. It shows that again cooperation is slightly better than no cooperation. However, there is a clear overlap of all the box-plots showing that difference between cooperating and standalone agents is not very significant. There is a small improvement if the number of agents is increased.

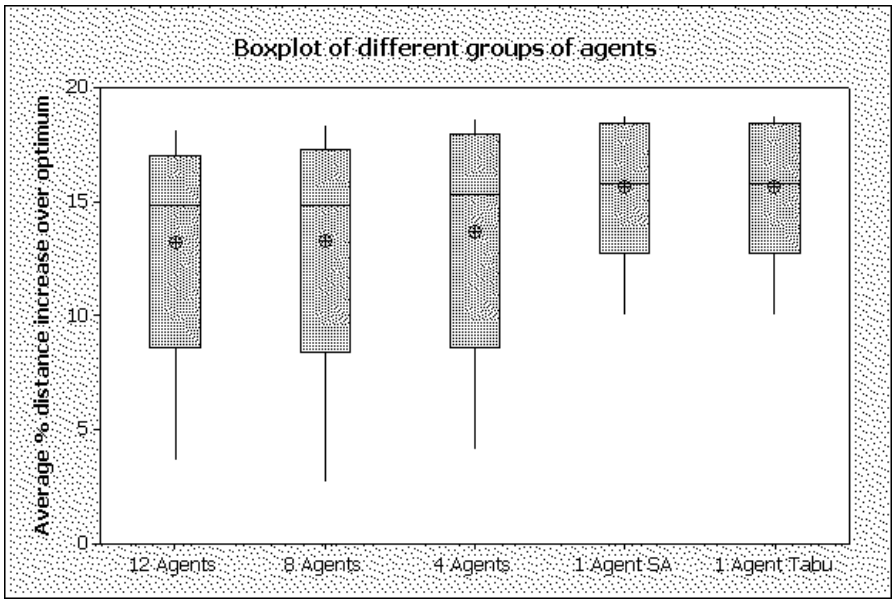


Figure 4: Cooperating agents versus no cooperation

7.3. Permutation flow-shop scheduling

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. The local search algorithm used by all the (meta-)heuristic agents is 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.

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 equations 5 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 2 shows this average percentage increase for each problem size for each agent grouping. The "1 agent" values are given for when the agent ran simulated annealing and when it ran tabu search as its (meta-)heuristic.

Table 2: 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	0.1228	0.1614	0.3048	20.0642	10.2630
20x10	0.8090	1.0016	1.0682	22.4359	12.8168
20x20	0.9041	0.9226	1.1304	17.0299	9.1332
50x5	0.0856	0.1205	0.1636	13.9210	5.3682
50x10	1.3764	1.4393	1.5416	22.7952	10.9588
50x20	3.3205	3.3880	3.4661	24.9155	13.0121
100x5	0.1383	0.1763	0.1763	10.8396	2.9393
100x10	0.6674	0.7599	0.7744	18.1045	6.5690
100x20	2.5287	2.6282	2.6777	23.0414	11.6709
200x10	0.4786	0.6071	0.6611	23.0414	4.3780
200x20	2.2395	2.3043	2.3181	21.6359	9.5804
500x20	1.5552	1.5640	1.6147	13.5150	4.9162

Table 2 shows clearly that there is a big difference between cooperating and non cooperating agents. It is significant 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 here the benefits of cooperation and compared to no cooperation are clear.

The spread of these results can be analysed more clearly using ANOVA. In figure 5 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.

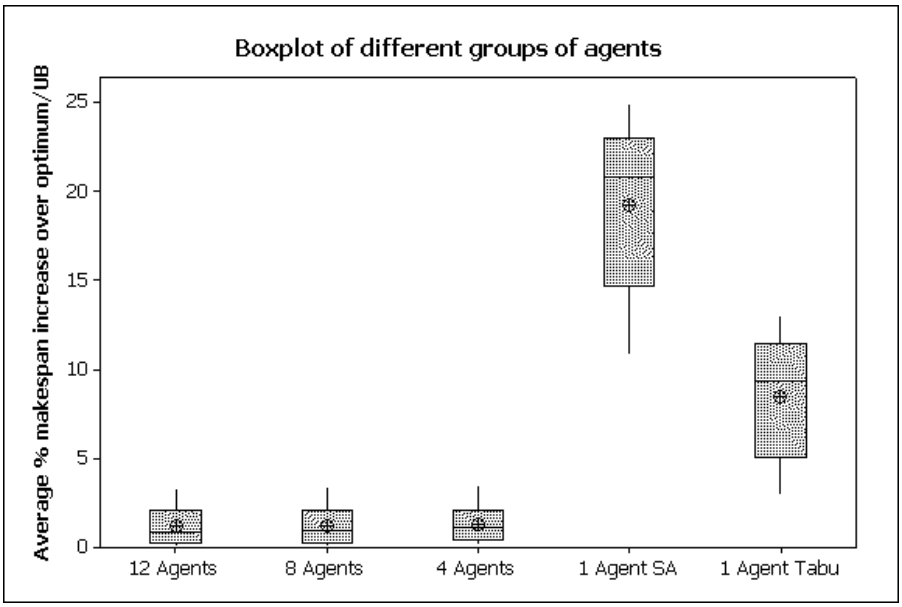


Figure 5: Cooperating agents versus no cooperation

The second test objective mentioned in section 5.2 above was to see if more agents cooperating produced better results than fewer agents cooperating. Figure 6 shows that while there is an overlap between the box plots of 12,8 and 4 agents cooperating, it is clear the 12 agents perform better than 8 and 8 agents are better than 4. There is a clear pattern emerging showing that more cooperating agents are better than fewer.

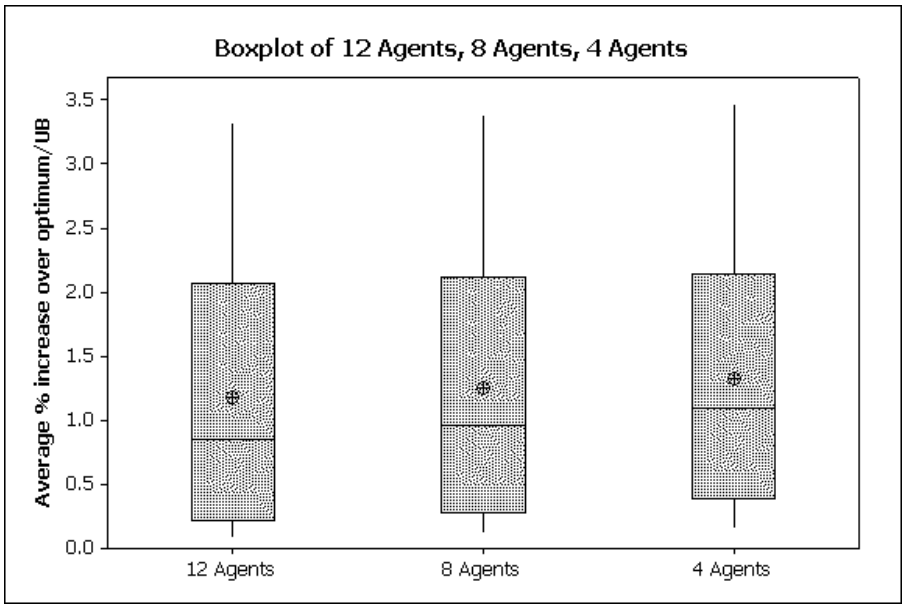
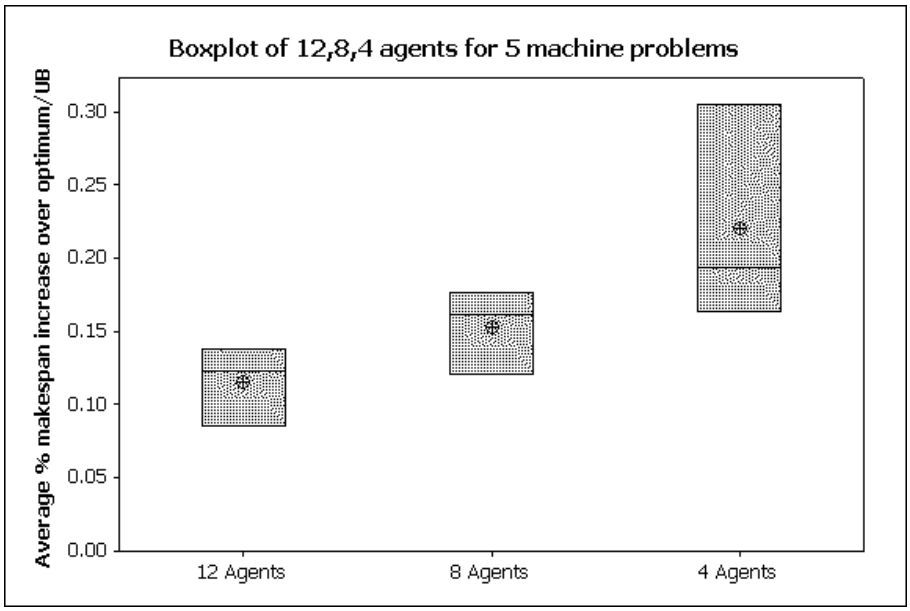
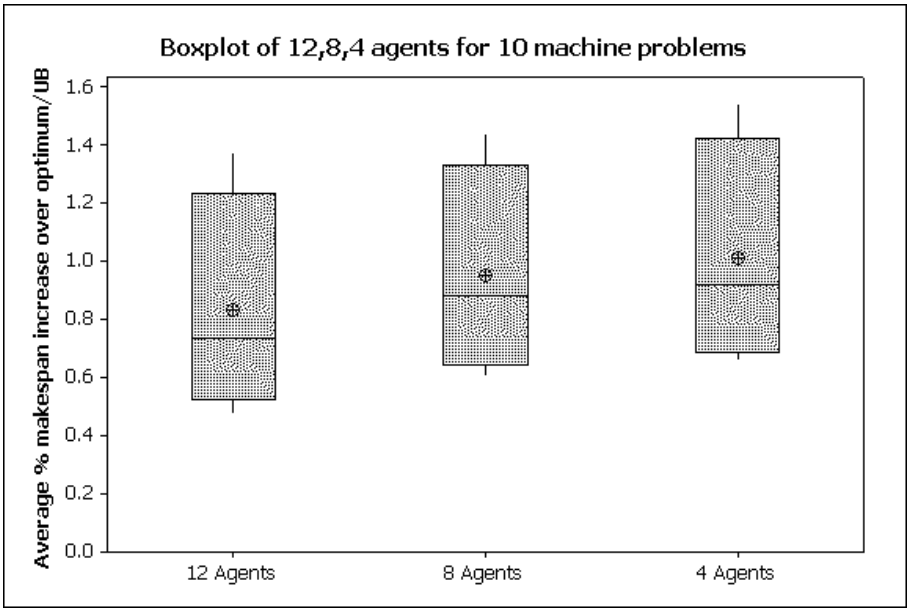


Figure 6: Comparison between 12,8 and 4 agents

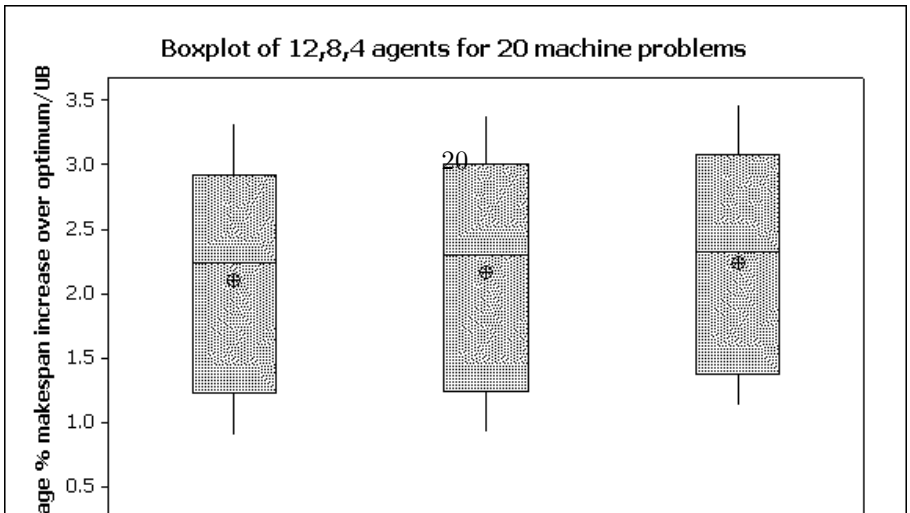
It can also be seen from figure 7 that the best results for the cooperating agents are when the system tries to solve 5 machine problems such as 20x5,50x5,100x5. But as the number of machines increases such as 20x10, 20x20, the system performs less well. This is phenomenon is also happens elsewhere in the literature. For example this can been seen in the results of Vallada and Ruiz (2009).



(a) 5 machine problems



(b) 10 machine problems



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 due to in the calculation of the makespan (equation 5)which takes 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. This would then make it harder to solve these types of problem as more possibilities have to be explored to find a good permutation with a good minimum makespan.

7.4. Diversification 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 the 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. Also depending whether the problem domain was PFSP or STSP the agents participated in 3002 or 5002 conversations respectively. Therefore t mechanism recorded the same number values for each agent.

Figure 8 is a line graph of a 4 agent scenario working on the first of Taillard's 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 was it is possible to compare the search path of each agent.

It it worth noting that while figure 8 charts path of the search of a 4 agent scenario for a PFSP, this system can just as easily for a STSP search. This mechanism is completely generic and works on any problem type and with any conversation scenario.

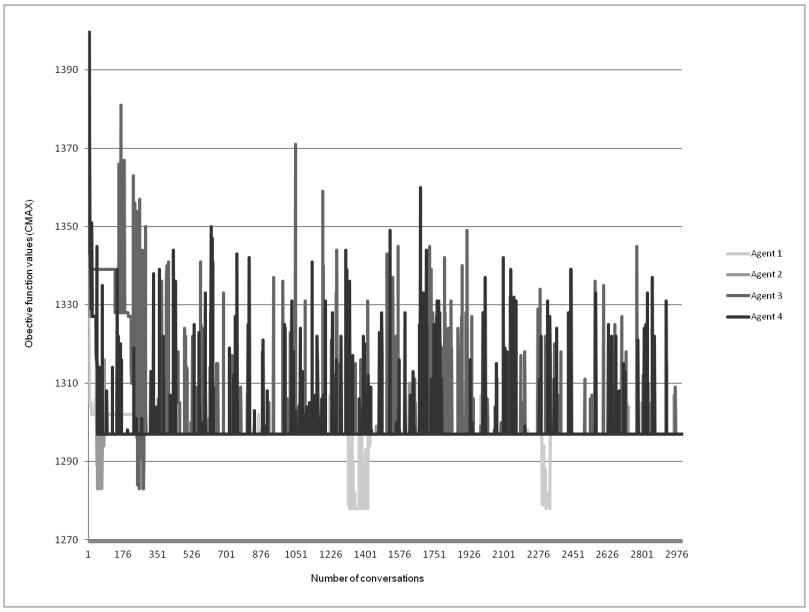


Figure 8: A graph of all the local minima or best results achieved by each agent

What is clear from figure 8 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 only gets down 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.

This is a very valuable information when designing heuristics. For example, it can be concluded that the system works well enough allowing new solutions to be built from finding good patterns. However, it can been that one agent (agent 4) seems to be holding back the search by contributing bad patterns. This information has been instrumental leading to new work on finding ways to recognise and penalise bad patterns.

8. Conclusions

The goal of the research presented in this paper is to propose a novel generic framework for cooperative search. The idea is to find cooperative protocols incorporating

pattern matching and reinforcement learning.

By conducting successful tests on benchmark problems on both PFSP and STSP, the results have showed that the framework is able to solve problems in different domains. This was achieved by the use of ontologies which enables the framework to be generic and be applied to different problem domains. These ontologies ensure that the meta-heuristic agents use and communicate the same objects when solving problems from different domains. Furthermore, the results show that cooperative search outperforms equivalent standalone (meta-)heuristics.

The results have also shown that cooperation with pattern matching only works well when there is a reinforcement learning facility included. This allows agents to exchange good patterns and to converge to good solutions. Without this feature, cooperation would diversify but would not necessarily converge.

The framework has a mechanism for charting diversification (section **??**, which so far has been used in an offline manner. Used this way it can aid parameter tuning and heuristic selection. In the future it is hoped it will be used online manner automatic heuristic selection and parameter tuning. This means the focus of this research and in the future is to develop a framework for automatic heuristic and parameter selection.

Finally, to try and meet the overall objective of developing a framework for automatic heuristic and parameter selection the authors have proposed an agent-based framework. There are many advantages to this. The agents are autonomous programs with their own internal representations of the problem they are trying to solve. Autonomy, means that the agents are flexible: they can use ontologies. As a consequence, they can run in many different configurations and work on many different problem types. It also means that they can be tailored to the computing resources available. It is easy to add or remove agents from the framework this can be done at run time. Autonomy also means that agents are able to alter their behaviour. This is feature the authors plan to develop with the robustness measure.

The work presented in this paper marks only the beginning what is a rich seam for research. Future research directions will include mechanisms to automatically tune parameters. With this in mind, it is intended that the ontology is extended to include parameter concepts. This will not only increase system flexibility but also enable us to start to automate parameter tuning allowing the (meta-)heuristic agents to detect and share good parameter settings. As a consequence a forecasting statistic will be developed which will be based on robustness data already collected by the agents. This will play a strong part in this work enabling the (meta-)heuristic agents to monitor the progress of the search (online) and offer suggestions to improve it.

There is an obvious limitation to the cooperation protocol. These are that bad patterns can occur just as frequently as good ones. This effect is ameliorated by using (meta-)heuristics to search further a new solution generated by the pattern exchange protocol. However this does not completely stop this drawback as an (meta-)heuristic agent starting from a worse position than the others can hold back others from converging to near optimal solutions.

One way to remedy this is to improve the pattern matching protocol is by enhancing the frequency scoring system. It is intended that an accumulated score will be added to each pair found in the frequency scoring process. In this way a value can be added to patterns that occur across conversations. Also bad patterns can be identified and marked making them tabu for a given number of conversations.

As the framework is completely generic, it intended to test the system further with algorithms written by third parties. In this way it will be possible to test current cooperation strategies on a number of new problems types. It is also planned to implement more cooperation strategies.

Finally, for the next body of work, the system will be applied to nurse rostering. The authors think this is an appropriate area of study because, not only can the platform be made more flexible, but also further use the agent property of autonomy. To this end an agent will be used to model groups of nurses either on a ward by ward basis or by requirements basis (part-time worker v full time works). One agent could represent the interests of each group and the system by negotiation and cooperation will find an optimal nurse rosters for these different interest groups. Also this project will also benefit from the improvements suggested above.

9. References

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

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

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

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

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.

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003. ISSN 0360-0300.

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

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

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, 1995. 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.

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

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

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

T. Hogg and C. P. Williams. Solving the really hard problems with cooperative search. pages 231–231, 1993.

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. ISSN 0377-2217.

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.

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.

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.

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, 1983.

D. Ouelhadj and S. Petrovic. A cooperative hyper-heuristic search framework. *Journal of Heuristics*, pages 1–23, 2010. ISSN 1381-1231.

M. Pinedo. Scheduling: theory, algorithms, and systems. 2002.

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.

Rubén Ruiz and Concepción Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494, 2005.

J. R. Searle. *Speech acts: An essay in the philosophy of language*. Cambridge university press, 1970. ISBN 052109626X.

F. C. N. I. P. Specification. At: http://www. fipa. org/specs/fipa00029. *SC00029H. pdf*.

E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64 (2):278–285, 1993.

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.

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.

E. Vallada and R. Ruiz. Cooperative metaheuristics for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 193(2):365–376, 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.

M. J. Wooldridge. *An introduction to multiagent systems*. Wiley, 2009. ISBN 0470519460.

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.