European Journal of Operational Research 254 (2016) 169–178

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Decision Support

A multi-agent based cooperative approach to scheduling and routing

Simon Martin[a,*], Djamila Ouelhadj[b], Patrick Beullens[c], Ender Ozcan[d], Angel A. Juan[e], Edmund K. Burke[f]

[a] Computational Heuristics Operational Research Decision Support (CHORDS) Group, University of Stirling, Department of Mathematics and Computer Science, UK
[b] Centre of Operational Research and Logistics, University of Portsmouth, Department of Mathematics, UK
[c] Mathematical Sciences and Southampton Business School and CORMSIS, University of Southampton, SO17 1BJ, UK
[d] Automated Scheduling, Optimisation and Planning Research Group, University of Nottingham, Department of Computer Science, UK
[e] Department of Computer Science – IN3, Open University of Catalonia, 156 Rambla Poblenou, Barcelona 08018, Spain
[f] School of Electronic Engineering and Computer Science, Queen Mary University of London, UK

ARTICLE INFO

Article history:
Received 10 March 2015
Accepted 28 February 2016
Available online 4 March 2016

Keywords:
Combinatorial optimization
Scheduling
Vehicle routing
Metaheuristics
Cooperative search

ABSTRACT

In this paper, we propose a general agent-based distributed framework where each agent is implementing a different metaheuristic/local search combination. Moreover, an agent continuously adapts itself during the search process using a direct cooperation protocol based on reinforcement learning and pattern matching. Good patterns that make up improving solutions are identified and shared by the agents. This agent-based system aims to provide a modular flexible framework to deal with a variety of different problem domains. We have evaluated the performance of this approach using the proposed framework which embodies a set of well known metaheuristics with different configurations as agents on two problem domains, Permutation Flow-shop Scheduling and Capacitated Vehicle Routing. The results show the success of the approach yielding three new best known results of the Capacitated Vehicle Routing benchmarks tested, whilst the results for Permutation Flow-shop Scheduling are commensurate with the best known values for all the benchmarks tested.

1. Introduction

Heuristics often come with a set of parameters, each requiring tuning for an improved performance. Moreover, different heuristics can perform well on different problem instances. Hence, there is a growing number of studies on more general methodologies which are applicable to different problem domains for tuning the parameters (Hutter, Babic, Hoos, & Hu, 2007; López-Ibánez, Dubois-Lacoste, Stützle, & Birattari, 2011; Ries & Beullens, 2015), generating or mixing/controlling heuristics (Burke et al., 2013; Ross, 2014). In this study, we take an alternative approach and use cooperating agents, where each agent is enabled to take a different method with different parameter settings.

By cooperative search we mean that (meta)heuristics, executed in parallel as agents, have the ability to share information at various points throughout a search. To this end, we propose a modular agent-based framework where the agents cooperate using a direct peer to peer asynchronous message passing protocol. An island model is used where each agent has its own representation of the search environment. Each agent is autonomous and can execute different metaheuristic/local search combinations with different parameter settings. Cooperation is based on the general strategies of pattern matching and reinforcement learning where the agents share partial solutions to enhance their overall performance.

The framework has the following additional characteristics. By using ontologies (see Section 3.2), we are aiming to provide a framework that is flexible enough to be used on more than one type of combinatorial optimisation problem with little or no parameter tuning. This is achieved by using our scheduling and routing ontology to translate target problems into an internal format that the agents can use to solve problems. So far, this approach has been applied successfully to Capacitated Vehicle Routing (CVRP), Permutation Flow shop Scheduling (PFSP), reported here and Nurse Rostering reported in Martin, Ouelhadj, Smet, Vanden Berghe, and Özcan (2013).

The aim of this study is to develop a modular framework for cooperative search that can be deployed, with little reconfiguration, to more than one type of problem. We also test whether interaction between (meta)heuristics leads to improved performance and

* Corresponding author. Tel.: +44 1786467462.
E-mail addresses: spm@cs.stir.ac.uk (S. Martin), djamila.ouelhadj@port.ac.uk (D. Ouelhadj), P.Beullens@soton.ac.uk (P. Beullens), Ender.Ozcan@nottingham.ac.uk (E. Ozcan), ajuanp@uoc.edu (A.A. Juan), e.burke@qmul.ac.uk (E.K. Burke).

http://dx.doi.org/10.1016/j.ejor.2016.02.045

if increasing the number of agents improves the overall solution quality.

1.1. Cooperative search in OR

The interest in cooperative search has risen due to successes in finding novel ways to combine search algorithms. Cooperative search can be performed by the exchange of states, solutions, subproblems, models, or search space characteristics. For a general introduction see, for example, Blum and Roli (2003), Clearwater, Hogg, and Huberman (1992), Crainic and Toulouse (2008), Hogg and Williams (1993), Talbi and Bachelet (2006) and Toulouse, Thulasiraman, and Glover (1999). Several frameworks have been proposed recently which incorporate metaheuristics such as Meignan, Creput, and Koukam (2008), Meignan, Koukam, and Créput (2010), Milano and Roli (2004) and Talbi and Bachelet (2006), or hyper-heuristics, as in Ouelhadj and Petrovic (2010). Also, El Hachemi, Crainic, Lahrichi, Rei, and Vidal (2014) explore a general agent-based framework for solution integration where distributed systems use different heuristics to decompose and then solve a problem.

In an effort to find ways to combine different metaheuristics in such a way that they cooperate with each other during their execution, a number of design choices have to be made. According to Crainic and Toulouse (2008) an *asynchronous* framework in particular could result in an improved search methodology; communication can then either be many-to-many (direct), where each metaheuristic communicates with every other, or it can be memory based (indirect), where information is sent to a pool that (other) metaheuristics can make use of as required.

Most cooperative search mechanisms in the OR literature deploy indirect communication through some central pool or adaptive memory. This can take the form of passing whole, or possibly partial, solutions, to the pool. Malek (2010), Milano and Roli (2004), Meignan et al. (2008, 2010) and Talbi and Bachelet (2006). Aydin and Fogarty (2004b) applied this approach to job shop scheduling. Recently, (Barbucha, 2014) has proposed an agent-based system for Vehicle Routing Problems where agents instantiate different metaheuristics which communicate through a shared pool.

Direct communication is used only in Vallada and Ruiz (2009) and Aydin and Fogarty (2004a), where whole solutions are passed from one process to another in an island model executing a genetic or an evolutionary simulated annealing algorithm respectively, and in Ouelhadj and Petrovic (2010), where a similar set-up is used for a hyper-heuristic. All three papers addressed the PFSP. Also, this approach is to an extent present in the evolutionary system of Xie and Liu (2009), who investigated the Travelling Salesman Problem. Kouider and Bouzouia (2012) propose a direct communication multi agent system for job shop scheduling where each agent is associated with a specific machine in a production facility. Here a problem is decomposed into several sub-problems by a "supervisor agent". These are passed to "resource agents" for execution and then passed back to the supervisor to build the global solution.

Little work has been done on asynchronous direct cooperation where partial solutions are rated and their parameters are communicated between autonomous agents all working on the total problem. So far, no direct cooperation strategy has been applied to more than one problem domain in combinatorial optimisation. To this end, the agents are truly autonomous and not synchronised. 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.

The outline for the rest of the paper is as follows. Section 2 provides formal problem statements for the two case studies. Section 3 describes the proposed modular multi-agent framework for cooperative search, while Section 4 describes how it is implemented. In Section 5 we discuss the experimental design. In Section 6 we report the results of the tests where, to the best of our knowledge, for three of the Capacitated Vehicle Routing instances we achieved better results than have been reported in the literature. Finally, Section 7 presents conclusions and suggestions for future work.

2. Test case problems

In this section we offer brief problem descriptions of the case studies applied to the agent-based framework proposed in this paper. We chose these instances as they are representative scheduling and routing problems. The algorithms instantiated by the framework are state-of-the-art implementations (Juan, Ruíz, Lourenço, Mateo, & Ionescu, 2010b; Juan, Faulin, Jorba, Caceres, & Marquès, 2013; Juan, Faulin, Ruiz, Barrios, & Caballé, 2010a; Juan, Lourenço, Mateo, Luo, & Castella, 2014). These are all examples of Simheuristics (Juan, Faulin, Grasman, Rabe, & Figueira, 2015). This makes them a good fit with the partial solutions identified by the system.

2.1. Permutation flow-shop scheduling problem

Let us assume that we have a set of n jobs, $J = \{1, ..., n\}$, available at a given time 0, and each to be processed on each of a set of m machines in the same order, $M = \{1, ..., m\}$. A job $j \in J$ requires a fixed but job-specific non-negative processing time $p_{j,i}$ on each machine $i \in M$. The objective of the PFSP is to minimise the *makespan*. That is, to minimise the completion time of the last job on the last machine C_{max} (Pinedo, 2002). A feasible schedule is hence uniquely represented by a permutation of the jobs. There are $n!$ possible permutations and the problem is NP-complete (Garey, Johnson, & Sethi, 1976).

A solution can hence be represented, uniquely, by a permutation $S = (\sigma_1, ..., \sigma_j, ..., \sigma_n)$, where $\sigma_j \in J$ indicates the job in the jth position. The completion time $C_{\sigma_j,i}$ of job σ_j on machine i can be calculated using the following formulae:

$$C_{\sigma_1,1} = p_{\sigma_1,1} \tag{1}$$

$$C_{\sigma_1,i} = C_{\sigma_1,i-1} + p_{\sigma_1,i}, \text{ where } i = 2, ..., m \tag{2}$$

$$C_{\sigma_j,i} = max(C_{\sigma_j,i-1}, C_{\sigma_{j-1},i}) + p_{\sigma_j,i}, \text{ where } i = 2, ..., m, \text{ and } j = 2, ..., n \tag{3}$$

$$C_{max} = C_{\sigma_n,m} \tag{4}$$

2.2. The Capacitated Vehicle Routing Problem

The Capacitated Vehicle Routing Problem (Dantzig & Ramser, 1959) can be defined in the following graph theoretic notation. Let $G(V, E)$ be an undirected complete graph where $V = \{v_0, v_1, v_2, ..., v_n\}$ is the vertex set and where E is a set of edges.

Let the set v_i (where $i = \{1, ..., n\}$) represent the customers who are expecting to be serviced with deliveries and let v_0 be the service depot. Also associated with each vertex v_j is a non-negative demand d_j. This value is given each time a delivery is made. For the depot v_0 there is a zero demand d_0.

The set E represents the set of roads that connect the customers to each other and the depot. Thus each edge $e \in E$ is defined as a pair of vertices (v_i, v_j). Associated with each edge is a cost $c_{i,j}$ of the route between the two vertices.

Finally there is also a set of unlimited trucks each with same loading capacity. The aim is to service all the customers visiting them once only and using as few trucks as possible. In any

potential delivery round a customer's demand has to be taken into account. The total demands of customers on the round must not exceed the capacity of the vehicle. This means that it is normally not possible to visit all customers with one truck. As a consequence each delivery round for a truck is called a *route*.

The goal of the CVRP problem is to minimise the overall travelling distance to service all customers with varying demand using a given number of trucks, each with the same fixed capacity.

This problem is NP-Hard Garey and Johnson (1979).

2.3. Benchmark instances

We used the following benchmark instances for testing the experiments described in Section 5. For PFSP, we selected 12 benchmark problems from Taillard (1993). Each Taillard PFSP benchmark instance is labelled as *taiX_j_m*, where *X* is the instance number and (*j*, *m*), where *j* indicates the number of jobs, and *m* the number of machines. In order to facilitate our analysis, we selected 12 of the harder instances as follows: two from the (50, 20) pool, two from the (100, 20) pool and then three from the (200, 10) and (200, 20) pools and finally three from the (500, 20) pool of instances for which an optimal solution is not known. For CVRP, we tested 12 problems from the benchmarks of Augerat et al. (1995). Each instance of this benchmark is denoted as *A-nM-kL*, where *M* and *L* indicate the number of delivery points including the depot and the target number of routes, respectively.

3. Agent-based framework

3.1. Framework architecture and operation

We describe a general agent-based distributed framework where each agent implements a different metaheuristic/local search combination. An agent continuously adapts itself during the search process using a cooperation protocol based on the retention partial solutions deemed as possible constituents of future good solutions. These are shared with the other agents.

The framework makes use of two types of agent: *launcher* and *metaheuristic* agents.

- The launcher agent is responsible for queueing the problem instances to be solved for a given domain, configuring the metaheuristic agents, successively passing a given problem instance to the metaheuristic agents and gathering the solutions from the metaheuristic agents. To achieve this it converts domain specific problem instances into the agent messaging protocol using an ontology for scheduling and routing (see Section 3.2). However the launcher agent plays no actual part in the search, its job is to prepare and schedule problems to be solved by the other agents.
- A metaheuristic agent executes one of the metaheuristic/local search heuristic combinations that are available. These combinations and their parameter settings are all defined on launching. In this way each agent is able to conduct searches using different combinations and parameter settings from the other agents employed in the search. Each metaheuristic agent conducts its search using the messaging structure defined in the ontology for scheduling and routing and uses no problem specific data and as such is generic.

A search proceeds with the launcher reading a number of problem instances into memory. It converts them into objects that can be defined by the Ontology for scheduling and routing (Section 3.2 below) and then sends each object, one at a time, to the metaheuristic agents to be addressed. For a given problem instance, the metaheuristic agents participate in a communication protocol which is in effect a distributed metaheuristic that enables them to search collectively for good quality solutions. This is a sequence of messages passed between the metaheuristic agents and each message is sent as a consequence of internal processing conducted by each agent. One iteration of this protocol is called a *conversation* and is based upon the well-known contract net protocol (FIPA, 2009). In order to arrive at a good solution the agents will conduct 10 such conversations.

To understand the pattern matching protocol it is necessary to explain the proposed model for scheduling and routing used throughout the framework.

3.2. Scheduling and routing ontology

The ontology (Gruber, 1993) plays an important role within our framework. It defines a set of general representational primitives that are used to model a number of scheduling and routing problems. The communication protocol and the heuristics are all based on data structures developed from these primitives. This means the framework is modular in that new (meta)heuristics can be easily developed and then deployed on different problems.

The ontology used by the framework generalises these notions as abstract objects.

- *SolutionElements:* A SolutionElement is an abstract object that can represent a problem specific object such as a *job* in PFSP or, a *customer or depot* in CVRP.
- *Edge:* An Edge object contains two SolutionElements objects. These are used to represent pairs of *jobs* or *customers* in a permutation that will be in the cooperation protocol to identify good patterns in improving permutations.
- *Constraints:* The Constraints interface is between the high level framework and the concrete constraints used by a specific problem. These are used to verify a valid permutation.
- *NodeList:* A NodeList object is a list of SolutionElements objects or Edges. It represents a *schedule of jobs* in the PFSP. In the case of CVRP, a NodeList represents a *Route* and is therefore a sublist of a full permutation.
- *SolutionData:* A SolutionData object is a list of NodeList objects and therefore is the permutation that is optimised by the framework. In this study it represents a *schedule of jobs* in PFSP, or a *collection of routes in CVRP*.

All message passing in the framework, including the whole ontology, is written in XML. This can be advantageous as many benchmark problems are also in XML making the interface between problem definition and ontology seamless in practice. Fig. 1 shows the structure of the ontology and how SolutionElements are the interface between the framework and a concrete problem.

3.3. Edge selection and short-term memory

The framework features a method of Edge selection and short-term memory. A conversation, as has been explained already, is a type of distributed heuristic. Its purpose is to identify constituent features of incumbent solutions that are likely to lead to the building of improving solutions.

This is achieved by using objects defined in the ontology. The solutionData object in the ontology is built from the sub-objects of NodeLists and Edges and SolutionElements. Thus, to represent a permutation of n jobs for PFSP, a SolutionData object is built from one NodeList object and which itself is made up $n-1$ Edge objects which are themselves built from n SolutionElements. Similarly a CVRP representation of n customers is one Solution Data object with x (this number is determined during the search) NodeLists. The NodeLists are built of $n-1$ Edges and n SolutionElements.

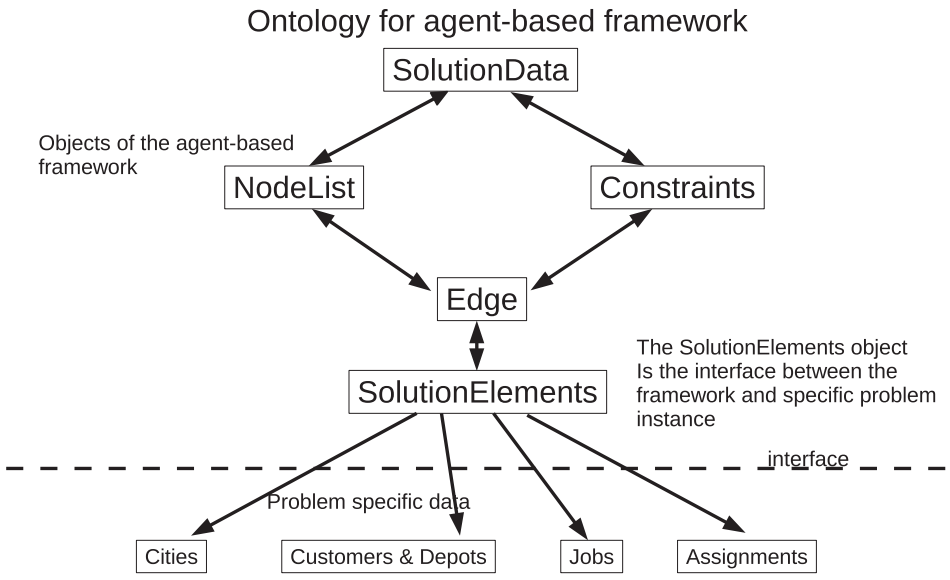


Fig. 1. The combinatorial optimisation ontology.

If we take a permutation of the unique ID numbers of each the SolutionElements objects we can represent a SolutionData object with 10 elements as follows: (3, 4, 6, 7, 5, 8, 9, 0, 1, 2). Furthermore we can break this permutation into a collection of Edge objects:

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

During a conversation, each agent runs its metaheuristic and produces a new incumbent solution. Each agent then breaks this solution into Edge objects and sends then to one of the metaheuristic agents that has been designated as the "initiator" for the duration of that conversation only. All metaheuristic agents are exactly the same and have the potential to take on the role of an initiator in a conversation.

The initiator agent collects all the Edge objects from all the other agents into a list and scores them by frequency. Here, frequency is the number of times an Edge appears in the initiators list. The only Edge objects that are retained are the ones that have the same score as the number of agents that are participating in the conversation. The idea here is that if an Edge occurs frequently in all incumbent solutions, it is likely to be an Edge that will be part of an improving solution. These retained good Edges are then shared by the initiator with the other agents.

Another feature is the learning mechanism where each agent keeps a short-term memory of good Edges. This is a queue of good Edges that operates somewhat like a Tabu list. An agent's queue is populated during the first conversation with edges from the incumbent solution produced by its metaheuristic. Thereafter the queue is maintained at a factor, that is 20 percent, of the size of the candidate solution for the problem instance at hand. In subsequent conversations, as new edges not already in the list arrive, they are pushed onto the front of the queue while other edges are removed from the back of the queue so that the size of the list does not change.

The Edges in the short-term memory are used at the start of each conversation to modify the performance of the agent's metaheuristic to enable it to find better solutions.

The basic idea of this learning mechanism is that both the RandNEH and RandCWS heuristics of Juan et al. (2015) used in this study make use of ordered lists to construct new solutions. These heuristics use biased random functions to choose items from these lists. We use the Edges identified by the learning mechanism to reorder these lists and so influence the way new solutions are constructed.

4. Implementation

The framework is implemented using JADE (Bellifemine, Caire, & Greenwood, 2007). It allows a developer to concentrate on the function and behaviour of agents while it handles inter-agent and inter-platform communication.

The configuration file of a launcher agent lists which problems are to be solved. It also contains how many conversations the metaheuristic agents are going to conduct for a particular problem.

At start-up, parameters determine which metaheuristic will be employed as well as any parameter settings associated with it. Once the metaheuristic agents have completed the set number of conversations they each send their best result to the launcher agent. The launcher then prints an output file with the best solution and objective function value.

The framework conducts a search where each agent is launched and registers with the JADE platform that hosts the framework. Once this is complete, the agents wait for the launcher agent to read in a problem from file. The launcher will then send the problem to each of the metaheuristic agents. Only when the metaheuristic agents receive that problem from the launcher do they embark on a search.

4.1. Heuristics used by the agents

In this study depending on whether they are solving PFSP or VRP, the agents instantiate the heuristics developed by Juan et al. (2010b, 2010a) respectively.

In the case of PFSP, the metaheuristic used is the Randomised NEH (RandNEH) algorithm of Juan et al. (2010b). It is a stochastic version of the classic heuristic of Nawaz, Enscore, and Ham (1983). Just as the NEH algorithm creates an ordered list of *jobs* sorted from tardiest to quickest, the RandNEH algorithm, instead of choosing jobs in order from the list, chooses them according to a randomised process based on the Triangular probability distribution.

While for the CVRP, the metaheuristic used is the Randomised Clarke Wright Savings (RandCWS) algorithm of Juan et al. (2010a). It is a stochastic version of the classic savings heuristic of Clarke and Wright (1964). Rather than generating new routes by choosing the greatest relevant saving from the *savings list*, it chooses according to a Geometric distribution where the *j*th *savings* from the list is chosen by a probabilistic function described in Juan et al. (2010a).

Both these algorithms have been integrated into our system according to our framework. This was quite a simple process where the heuristics implement the abstract objects defined in the scheduling and routing ontology. For example, the Edge and Job objects of the RandNEH algorithm are now subclasses of the Edge and SolutionElements abstract classes of the framework. Similarly for VRP problems, where the Route, Edge and Customer objects are now subclasses of the NodeElements, Edge and SolutionElements objects of the framework.

This means we can use the *good Edges* found as a result of a *conversation* of the framework to modify the Job lists and Saving lists of the RandNEH and RandCWS algorithms respectively.

In the case of PFSP, the list of Edges found by the agents is turned into a list of SolutionElements (Jobs) where their order in the Edge list is preserved. The Jobs list generated by the RandNEH algorithm is then reordered with respect to the list of Jobs generated from the Edge list, with the new Jobs being moved to the front of the list. This affects the operation of the RandNEH algorithm where the new Jobs are likely be favoured in the construction of any new improving schedule.

It is a similar process for the RandCWS algorithm. However, this time the Edges in Edge list are also Super Classes of the Edges in the Savings List. Again, the Savings List is reordered with respect to the Edge list where these Edges are moved to the head of the Savings List. This again affects the operation of the RandCWS algorithm favouring the *good* Edges found as a result of the Agents' conversations.

4.2. Description of a conversation

Fig. 2 shows the edge selection protocol used by the metaheuristic agents. One complete execution of the algorithm illustrated is a *conversation*. In any conversation, there will be an agent that takes on the role of an initiator and the others are responders. In the very first conversation agent1 will always take on the role of initiator. Thereafter, any agent can be the initiator, but it is determined in the previous conversation which agent will be the initiator for the current conversation (see below).

In Fig. 2, an agent taking on the role of initiator starts a conversation. At the start of a conversation, each agent either takes a list of Edge objects generated from a previous conversation or from one generated by the launch agent (see I_1 and R_1 in Fig. 2).

The agents then find new incumbent solutions using their given heuristics in conjunction with the edges provided in the previous step (see I_2 and R_2 in Fig. 2).

The initiator breaks its incumbent solution into edges and then invites the responder agents to do the same and send them to the initiator, I_3 and R_3 of Fig. 2.

The receiving agents also send the value of their best-so-far solution. This will be used by the initiator to determine which agent will be the new initiator in the next conversation (see I_4 in Fig. 2).

In I_4, the initiator receives the Edge objects from the responding agents and collects them together. Each Edge object is scored and ranked based on frequency. This can be seen in box I_4 of Fig. 2 as the function *getScore*.

In I_4 of Fig. 2, through the function *getInitiator*, the initiator also determines which metaheuristic agent is going to be the initiator in the next conversation. This is achieved by choosing the best objective function value to be the initiator. In the case of a tie, the agent is chosen arbitrarily from these values.

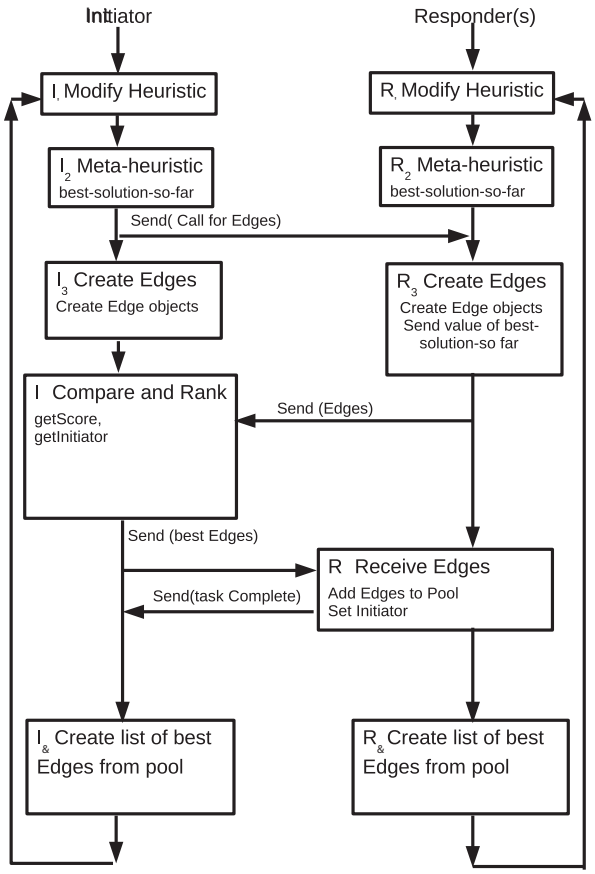


Fig. 2. The cooperation protocol showing one iteration of a conversation.

The initiator then sends good *Edge* objects, found during this conversation, to the receiving metaheuristic agents.

Each agent keeps a pool or short-term memory of high scoring Edge objects. The pool acts as a sort of queue and its length is set when the agent is launched. In this study all the agents have a pool size of 20 percent of length of the instance currently being addressed. During the first conversation, each agent populates its pool as good edges are identified. Once the pool is up to size, it is maintained as a queue as described in Section 3.3.

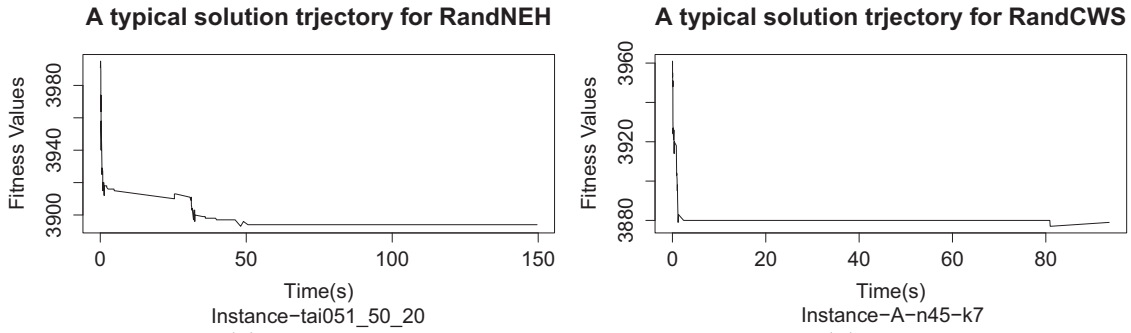
The other metaheuristic agents receive the lists of Edge objects from the initiator (see box R_4 in Fig. 2). They also update their internal memory's or pools as described above. In box I_5 and R_5 of Fig. 2, both initiator and responder metaheuristic agents then each create a new solution by using edges from their updated internal pools. These good edges are passed to the metaheuristic the agent is configured to execute in the current search. The metaheuristic uses these good edges when it is next called at the start of the next conversation (back to I_1 and R_1 of Fig. 2). This process repeats and continues until the number of conversations set from the launcher agent are completed.

5. Experimental design

In this section we discuss the experimental design.

5.1. Launcher agent

One launcher agent is invoked in each run. The launcher agent reads from a configuration file the number of agents to be



(a) tai051-50-20 (b) A-n45-k7

Fig. 3. Typical Solution trajectories of the RandNEH and RandCWS algorithms.

instantiated (see Section 5.4) as well as the number of conversations that will be conducted during the test.

The launcher agent executes a construction heuristic to build an initial solution for each instance and run: for PFSP a biased-randomised version of the NEH algorithm (Nawaz et al., 1983) with Taillard's speedups implemented by Juan et al. (2010b); and for CVRP, the Randomised CW Savings algorithm (Juan et al., 2010a; Juan et al., 2014). This initial solution is passed on to each of the individual agents.

5.2. The number of conversations

Using a standard computer, Juan et al. (2010a, 2010b); Juan et al. (2014) noticed that the RandNEH and RandCWS heuristics were able to provide near-optimal solutions for most instances in a maximum time of about 2.5 minutes. We benchmarked their code using a similar computer configuration and observed the same phenomena. Therefore, we decided to use this running time as a maximum-allowed computing time during our experiments, which were run in a more powerful computing environment.

This gave us a guide as to how long our system should be run and therefore determine the number of conversations that would be needed. The time taken for the agents to complete a conversation is mainly governed by the time taken for an agent's given heuristic to execute. To this end, we conducted tests showing that both heuristics typically have a period of maximum improvement of about 12 seconds. As an example, Fig. 3 plots the solution trajectories of the PFSP instance tai051 and the CVRP instance A-n45-k9 against time. We can see that these algorithms have their period of greatest improvement in about the first 12 seconds of operation. Thus we determined that the system should execute 10 conversations for our system to run for about the same time as the standalone versions of the RandNEH and RandCWS heuristics. This would also take into account any lag caused by the asynchronous nature of the system.

5.3. Parameter settings

Since the RandCWS and RandNEH methods of Juan et al. were already written in JAVA, they were integrated with minimum effort as a module of our agent based system. They utilise the edge selection heuristic of the agent-based system by taking edges identified during each conversation and re-ordering the jobs list of the RandNEH algorithms and the savings list of the RandCWS algorithm as explained in Section 4.1.

Both algorithms use a random seed which is a number that introduces a bias to a random number generator. In the tests for both the PFSP and CVRP, each agent is configured with exactly the same random seeds (Juan et al., 2010a, 2010b).

However, in their article (Juan et al., 2011) describe how they combined Monte-Carlo simulation techniques with the Clarke Wright Savings algorithm to develop the probabilistic RandCWS algorithm. It was designed so that it would require little parameter tuning. To this end, they describe a parameter α that is used to define different geometric distributions. Such a distribution can then be used by the RandCWS heuristic to choose the next edge from the Clarke Wright Savings list as part of its solution building process. The α-parameter is itself chosen at random from a uniform distribution between two values (a, b) where $0 < a \leq b < 1$. In their paper, Juan et al choose α-values from the interval $\alpha \in \{0.05 - 0.25\}$. They show that for any α-value in this interval, the algorithm will give similar and good performance. In correspondence with the authors, it was confirmed that the algorithm will perform less well for α-values of above 2.3, while at the other end of the range α-values close to the 0.05 will perform as any in the cited interval.

The intuitive idea for spreading the *α-values* is to maximise the use of different distributions during a search. While these choices do not affect the solution quality it means the agents will produce slightly different solutions which will produce different edges that will enhance the performance of the distributed edge selection algorithm.

In both case studies each metaheuristic is allowed to run for 12 seconds each time it is called.

Following (Juan et al., 2013; Juan et al., 2014) in what we call our standalone experiments (that is the traditional case without cooperative search being used) we compare our cooperating agents with the standalone by running the experiments for each group for a maximum time of 40 minutes to match the computational effort of the system running 16 agents i.e. 16×150 seconds $=$ 2400 seconds (40 minutes). Thus all agents vs standalone comparisons are made against this worst case scenario.

5.4. Experimental set-up

The main hypothesis to be tested in these experiments is that cooperating agents produce better results than their standalone equivalents. The results are also compared with state-of-the-art results for each of these benchmarks. To this end, for each instance of the tests the following scenarios were run:

The CVRP tests were conducted as follows with *α-values* selected on 0.01 increments from the set {0.03 to 0.18}

- Standalone agent: 1 metaheuristic agent where the *α-value* $=$ 0.03
- 4 agents: $\alpha \in \{0.03 - 0.06\}$
- 8 agents: $\alpha \in \{0.03 - 0.1\}$
- 12 agents: $\alpha \in \{0.03 - 0.14\}$
- 16 agents:$\alpha \in \{0.03 - 0.18\}$

Table 1
The average (avr.) and best percentage deviation from the upper bound over 20 runs for each instance for PFSP. The best values are highlighted in bold.

Instance	BKS	Zobolas et al.	1 agent		4 agents		8 agents		12 agents		16 agents	
		(percent)	Avr. (percent)	Best (percent)	Avr. (percent)	Best (percent)	Avr. (percent)	Best (percent)	Avr. (percent)	Best (percent)	Avr. (percent)	Best (percent)
tai051_50_20	3850	0.77	0.92	**0.39**	0.84	0.55	0.76	0.47	0.69	**0.39**	0.63	0.44
tai055_50_20	3610	1.03	0.54	0.44	0.67	0.50	0.62	**0.28**	0.57	0.36	0.50	0.30
tai081_100_20	6202	1.63	1.55	1.23	1.52	1.26	1.41	1.06	1.34	1.03	1.30	**1.02**
tai085_100_20	6314	1.57	1.39	1.00	1.34	1.11	1.22	0.97	1.15	1.00	1.11	**0.89**
tai091_200_10	10,862	0.24	0.12	**0.09**	0.09	**0.09**	0.09	**0.09**	0.09	**0.09**	0.09	**0.09**
tai095_200_10	10,524	**0.03**	0.10	**0.03**	0.09	**0.03**	0.05	**0.03**	0.03	**0.03**	0.03	**0.03**
tai101_200_20	11,195	1.34	1.49	1.30	1.38	1.09	1.25	1.01	1.22	1.06	1.19	**0.93**
tai105_200_20	11,259	1.04	1.08	**0.70**	1.02	0.89	0.94	0.78	0.94	0.83	0.88	0.71
tai106_200_20	11,176	1.11	1.60	**1.25**	1.55	1.35	1.44	1.27	1.43	**1.25**	1.42	1.33
tai111_500_20	26,059	0.73	0.99	0.74	1.01	0.88	0.95	0.86	0.92	0.87	0.88	**0.69**
tai115_500_20	26,334	0.82	0.99	0.74	1.01	0.88	0.95	0.86	0.92	0.87	0.88	**0.69**
tai116_500_20	26,477	0.49	0.72	0.56	0.69	0.56	0.67	0.60	0.62	0.57	0.61	**0.54**

Table 2
Table showing cooperating agents performing better than the standalone equivalent at the 95% in PFSP.

Instance	4 vs 1	8 vs 1	12 vs 1	16 vs 1
tai051_50_20	$\geq$	$>$	$>$	$>$
tai055_50_20	$\leq$	$\leq$	$\leq$	$\geq$
tai081_100_20	$\geq$	$>$	$>$	$>$
tai085_100_20	$\geq$	$>$	$>$	$>$
tai091_200_10	$>$	$>$	$>$	$>$
tai095_200_10	$\geq$	$>$	$>$	$>$
tai101_200_20	$>$	$>$	$>$	$>$
tai105_200_20	$\geq$	$>$	$>$	$>$
tai106_200_20	$\geq$	$>$	$>$	$>$
tai111_500_20	$\leq$	$\geq$	$>$	$>$
tai115_500_20	$\leq$	$>$	$>$	$>$
tai116_500_20	$\geq$	$>$	$>$	$>$

Table 3
Table showing different groups cooperating agents perform at the 95 percent confidence level in PFSP.

Instance	8 vs 4	12 vs 8	16 vs 12	16 vs 8	16 vs 4
tai051_50_20	$>$	$\geq$	$\geq$	$>$	$>$
tai055_50_20	$>$	$\geq$	$>$	$>$	$>$
tai081_100_20	$>$	$\geq$	$\geq$	$>$	$>$
tai085_100_20	$>$	$\geq$	$\geq$	$>$	$>$
tai091_200_10	$\approx$	$\approx$	$\approx$	$\approx$	$\geq$
tai095_200_10	$>$	$\geq$	$\geq$	$\approx$	$>$
tai101_200_20	$>$	$\geq$	$\geq$	$\geq$	$>$
tai105_200_20	$>$	$\leq$	$>$	$>$	$>$
tai106_200_20	$>$	$\geq$	$\geq$	$\geq$	$>$
tai111_500_20	$>$	$>$	$\geq$	$>$	$>$
tai115_500_20	$>$	$\geq$	$\geq$	$\geq$	$>$
tai116_500_20	$>$	$>$	$\geq$	$>$	$>$

The PFSP tests were conducted similarly but without the need for α-*values*.

They were tested in this way so that standalone agents running just one metaheuristic at a time can be compared statistically with groups of cooperating agents in order to test the main hypothesis.

Every instance was tested 20 times. The resulting values are then used to evaluate the performance of the test. In particular the average and minimum value of the 20 runs for each problem are taken. These are compared with the known optimal or best values for each problem instance.

To test the hypothesis that agents cooperating by edge selection perform better than standalone agents, Wilcoxon signed rank tests are conducted for each benchmark instance, with a 95 percent confidence level. We used the Wilcoxon test rather than *t*-test because we cannot guarantee that the test results will be normally distributed (Moore & McCabe, 1989). These tests compare the difference between the distributions of 16, 12, 8, and 4 agents cooperating with the standalone agents. A secondary hypothesis is explored where the performances of groups of 4, 8, 12 and 16 agents are compared using the Wilcoxon signed rank test to ascertain whether increasing the number of agents results in better performance. The following notation is used in Tables 2, 3, 6 and 7. Given two algorithms (or different settings for the same algorithm); A vs B, $>$ ($<$) denotes that A (B) is better than B (A) and this performance difference is statistically significant at a 95 percent confidence level. However, $\geq$ ($\leq$) denotes that A (B) is better than B (A) although statistical significance could not be supported. Lastly, $\approx$ denotes the case where both approaches consistently achieve the same value.

The results for each problem are averaged and the average percentage deviation from the known optimum is calculated. The percentage deviation from a known optimum is calculated in the standard manner:

$$\frac{Method_{solution} - Best_{solution}}{Best_{solution}} \times 100 \tag{5}$$

The results are also analysed to find the best result of each group of agents over the 20 runs of each problem instance (Juan et al., 2013; Juan et al., 2014).

5.5. Machines

All tests were run on the same Linux cluster using 8 identical machines; two agents were run per-node of the cluster. The agents were configured to use 2 gigabyte of memory.

6. Results of experiments

6.1. Permutation flow-shop scheduling results

Table 1 shows the average percentage deviation from the best known or optimum value for each of the benchmark instances tested, as well as the percentage deviation for the best value found across the 20 runs. The table also compares our results with the Hybrid Genetic algorithm of Zobolas, Tarantilis, and Ioannou (2009). Here the average value reported by Zobolas et al. (2009) is given as a percentage deviation from the best known solution. Despite the fact that this is a type of hyper-heuristic system where the only parameter tuning is the number of conversations executed, the PFSP results are competitive with the state-of-the-art results for these problem instances. It is only in the larger three instances where our average deviation is not better than that of Zobolas et al. (2009).

Table 4
Patterns found by 4 cooperating agents PFSP for problem tai051_50_20.

Agents	*Edges*
agent1	(14,15) (4,25) (32,22) (39,16) (25,50) (19,41) (13,32) (44,45) (45,6) (50,3) (28,38)
agent2	(35,34) (9,30) (5,10) (2,44) (12,37) (1,7) (4,50) (10,1) (24,42) (50,3)
agent3	(3,12) (37,39) (30,46) (50,3) (35,15) (41,7) (34,33) (38,24) (47,23) (42,49)
agent4	(40,21) **(22,13)** (6,42) (33,40) (26,2) (5,14) (7,18) (37,28) **(39,35)** (44,11)

With respect to answering our main hypothesis: "is cooperation by pattern matching better than no cooperation?", we compared 4 agents cooperating against a standalone agent (see Section 5.3). In addition, we wanted to test if increasing the number of agents produced a statistically significant improvement in the results. Tables 2 and 3 list these results; in each case we tested for statistical significance.

In Table 2, with the exception of the *tai*055_50_20 instance, it can be seen that groups of 812 and 16 agents perform better than the standalone with statistical significance. However, for the *tai*055_50_20 instance, 16 agents show some improvement, if not statistically, over the standalone. Furthermore, two instances of 4 agents perform statistically better than the standalone but the rest all show some improvement but not at the 95 percent level.

Table 3 explores the possibility that adding more agents leads to better results. Here we can see that 8 agents perform statistically better than 4, while 12 agents show some improvement, but not statistically, over 8. The same is true for 16 over 12 agents. However the instances tai091_200_10 and tai105_200_20 achieve statistical significance as well. By the time we get to 16 vs 4 agents, 16 agents always perform statistically better except for tai091_200_10 where statistical significance is not reached. It should also be noted for tai091_200_10 while the cooperating agents perform better than the standalone, thereafter they all achieve the same value. It is clear that progressively increasing the number of agents from 4 to 8 to 12 to 16 results in an increase in performance. However this improvement is not always statistically significant. If we consider the column of the table where 16 agents are compared with 8 we see that the level of improvement gains more significance. This is suggestive that it is better to increase the number of agents by a factor of 2.

The cooperation mechanism used in this study works by identifying and sharing good patterns that form partial solutions to the problem at hand. These are then passed to a metaheuristic to build a new putative solution to the problem. Given this, it is interesting to study the patterns (edges) identified by each agent and compare them to the final solution found by the system. To this end, the final permutation (Edges which appear in the final solution and are identified during the search (see Table 4)are highlighted in bold) < **12**, 37, 20, 31, **39**, **35**, 34, 6, 40, 5, 10, 1, 7, 15, 33, 43, 24, 42, 27, 29, 46, 47, 36, 23, 14, 2, 44, 8, 45, 17, **13**, **22**, 21, 48, 18, 28, 16, 49, 38, 19, 26, 41, 11, 32, 25, 9, 30, 4, **50**, **3** > of jobs found by the system during one run of the tai051_50_20 instance is compared with the patterns in Table 4. These are all the unique edges identified during this search. These edges are identified multiple times but the table only shows them once.

Indeed some edges (highlighted in bold) identified by the system do end up in the final job permutation. Furthermore, we can identify linked edges such as 50, 3, 12 at the end and beginning of the permutation. However, these are not as many as seen with CVRP results below because of the way the makespan 4 is calculated as a special cumulative sum of columns of jobs.

Table 6
Table showing cooperating agents performing better than the standalone equivalent at the 95 percent in CVRP.

Instance	4 vs 1	8 vs 1	12 vs 1	16 vs 1
A-n38-k5	≦	>	>	>
A-n39-k6	≦	≧	≧	≧
A-n44-k7	≦	>	>	>
A-n45-k6	≧	>	>	>
A-n45-k7	≦	≧	≧	>
A-n55-k9	≦	>	>	>
A-n60-k9	≦	>	>	>
A-n61-k9	≧	>	>	>
A-n62-k8	≦	>	≧	>
A-n63-k9	≦	≧	>	>
A-n65-k9	≧	>	>	>
A-n80-k10	≦	>	>	>

6.2. Capacitated Vehicle Routing results

Table 5 compares the percentage deviation for average and best results for the different groups of agents from the best known solution. The table also compares our percentage deviations for these problem instances with those of Altınel and Öncan (2005) (donated by A) and Juan et al. (2010b) (denoted by B). However, Juan et al. (2010b) only has results for a selection of the instances we tested. They represent the latest work on these benchmark instances so we have included them for comparison. Comparing our results with those of Altınel and Öncan (2005) and Juan et al. (2010b) we can see that agents improve on their results. Furthermore, to the best of our knowledge, in four cases we have found results that are better than the current best known solutions. *A-n*39-*k*6,*A-n*45-*k*7,*A-n*55-*k*9 and *A-n*63-*k*9 are highlighted in

Table 5
The average (avr.) and best percentage deviation from the optimum/upper bound over 20 runs for each instance for CVRP.

Instance	BKS	A and B	Juan et al.	1 agent		4 agents		8 agents		12 agents		16 agents	
		(percent)	(percent)	Avr. (percent)	Best (percent)	Avr. (percent)	Best (percent)	Avr. (percent)	Best (percent)	Avr. (percent)	Best (percent)	Avr. (percent)	Best (percent)
A-n38-k5	734.18	3.577	0.54	0.07	0.04	0.09	0.04	0.02	−0.03	−0.02	−0.03	*−0.03*	−0.03
A-n39-k6	833.14	2.233	–	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
A-n44-k6	939.33	2.394	–	0.63	0.57	0.70	0.57	0.55	0.39	0.40	0.29	0.32	−0.12
A-n45-k6	944.88	1.383	–	0.92	0.92	0.92	0.92	0.69	0.00	0.20	0.00	0.00	0.00
A-n45-k7	1147.28	1.842	0.07	0.05	−0.03	0.07	−0.02	0.03	−0.03	0.03	−0.03	*−0.01*	−0.48
A-n55-k9	1074.46	2.378	0.14	0.13	0.05	0.26	0.05	0.06	0.05	0.05	0.05	0.05	0.05
A-n60-k9	1355.80	1.64	0.13	0.50	0.50	0.50	0.50	0.46	0.22	0.40	0.22	0.37	0.22
A-n61-k9	1039.08	1.654	0.49	0.27	0.26	0.26	0.26	0.25	0.13	0.23	0.12	0.22	0.12
A-n62-k8	1294.28	4.648	–	0.70	0.62	0.76	0.62	0.62	0.62	0.65	0.62	0.62	0.62
A-n63-k9	1619.90	2.051	–	0.75	0.45	0.88	0.69	0.73	0.40	0.53	0.14	0.32	0.14
A-n65-k9	1181.69	2.392	0.66	1.06	1.05	1.05	0.72	0.92	0.28	0.82	0.64	0.61	0.14
A-n80-k10	1766.50	2.952	0.2	1.04	0.99	1.04	0.99	0.98	0.77	0.87	0.77	0.85	0.70

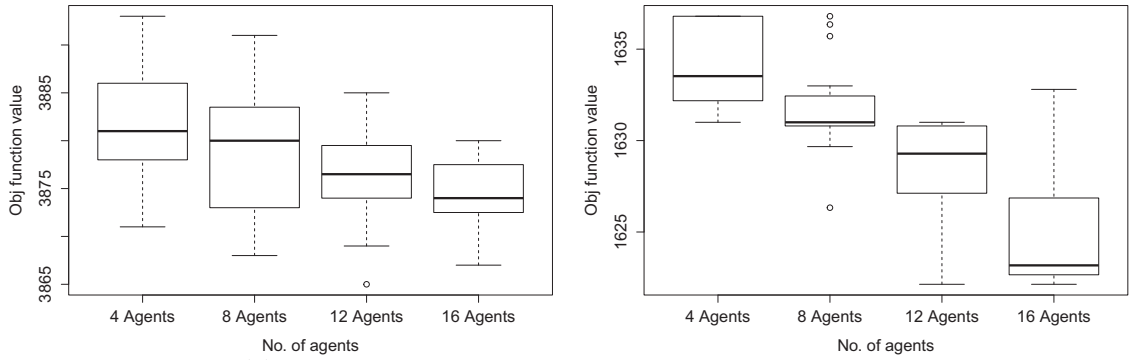


Fig. 4. Boxplots of objective values obtained in 10 runs for 16, 12, 8 and 4 agents on a selected instance from the (a) STSP, (b) PFSP, and (c) CVRP problem domains.

Table 7
Table showing different groups cooperating agents perform at the 95 percent confidence level in CVRP.

Instance	8 vs 4	12 vs 8	16 vs 12	16 vs 8	16 vs 4
A-n38-k5	>	>	>	>	>
A-n39-k6	≥	≥	≥	≥	≥
A-n44-k7	>	>	>	>	>
A-n45-k6	>	>	>	>	>
A-n45-k7	>	≥	≥	≥	>
A-n55-k9	>	≥	≥	≥	>
A-n60-k9	>	≥	≥	>	>
A-n61-k9	>	≥	≥	≥	>
A-n62-k8	>	≤	≥	≥	>
A-n63-k9	>	>	>	>	>
A-n65-k9	>	≥	>	>	>
A-n80-k10	>	>	≥	>	>

Table 8
Final Solution to CVRP problem A-n38-k5.

Route name	Routes
Route1	[1, 8, 6, 12, 28, 23, 33, 1]
Route2	[1, 27, 13, 4, 2, 5, 17, 26, 7, 30, 1]
Route3	[1, 9, 34, 36, 24, 31, 11, 22, 1]
Route4	[1, 10, 18, 37, 14, 16, 3, 15, 25, 1]
Route5	[**1**, 21, **38, 32, 29, 35, 20**, 19, 1]

Table 9
Patterns found by 4 cooperating agents for CVRP problem A-n38-k5.

Agents	*Edges*
agent1	**(35,20)** (38,32) (29,35) (20,19) (21,38) (32,29) (1,21) (19,1)
agent2	(30,31) (11,1) **(1,19)** (31,11) (35,30) (19,35)
agent3	**(35,20)** (29,19) (1,21) **(20,1)** (32,29) (38,32) (21,38) (19,35) (1,19)
agent4	**(19,1) (32,38) (38,29) (35,20)** (21,32) (20,19) (29,35)

italics for the best average value and in best for our best overall score.

Again we tested for the main hypothesis. We compared groups of 4, 8, 12, and 16 agents cooperating against a standalone agent. As before, we tested for statistical significance using the Wilcoxon signed rank test at the 95 percent confidence level. Table 6 lists these result using the same notation as used in Table 2 above. As with the PFSP, 4 agents cooperating do not show any improvement from their standalone equivalent. However, groups of 8, 12 and 16 agents with increasing certainty perform better than the standalone agent. Indeed 16 agents all perform better a 95 percent confidence level except for the *A-n39-k6* instance.

In Table 7, we report the results of our tests for the secondary hypothesis. As with the PFSP results, we can see a gradual improvement as more agents are added. However, again it seems it is necessary to double the number of agents each time in order to observe improvement in results. The addition of 4 agents each time results in an improvement that is not always statistically significant. However, if the agents are doubled each time in groups of 4, 8 and 16 there is a greater proportion of statistically significant improvement from the additive case.

Finally, we show the patterns generated for a sample on problem instance *A-n38-k5* in Table 9 and compare them to the final result of this run in Table 8. We highlight in bold those edges identified by the agents in Table 9 that end up in the final solution in Table 8. As can be seen there are many more such edges than for the PFSP. This is because the relationship between edges and cities is much more direct in the case of CVRP as costs are calculated as 2D-euclidean distances between cities.

From this study, we conclude that with no parameter tuning between case studies our system can produce results which are commensurate with the state-of-the-art studies in both fields. Furthermore, in four instances with the CVRP tests we were able to the best of our knowledge beat the current best results for these instances. We were also able to show for groups of 8, 12 and 16 agents compared with the standalone equivalent, that cooperation by pattern finding is better than no cooperation. Finally, we are also able to show that doubling the number agents each time leads to improving results as shown in Fig. 4.

7. Conclusion

In this paper we proposed a general agent-based distributed framework where each agent implements a different metaheuristic/local search combination. An agent continuously adapts itself during the search process using a cooperation protocol based on reinforcement learning and pattern finding. Good patterns that make up improving solutions are identified by frequency of occurrence in a conversation and shared with the other agents. The framework has been tested on well known benchmark problems for two tests cases PFSP and CVRP. In both cases, with no parameter tuning between domains, the platform performed at least as well as the state-of-art. For CVRP, we were able, in cases of *A-n38-k5*, *A-n44-k6* and *A-n45-k7* to improve on the best known solutions for these instances.[1]

We have also shown eight or more agents perform better than a standalone agent with a 95 percent confidence level. Furthermore, we have shown with a reasonable level or certainty, if not always with 95 percent confidence, that an improvement in performance can be achieved each time you double the number, up to 16, agents used.

[1] http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/.

The distributed computing framework presented can be run on a local network of personal computers each using 2 gigabyte memory.

The framework also aims to be generic and modular, needing very little parameter tuning across different problem types tested so far. It has been applied successfully to PFSP and CVRP. It has also been used to model fairness in Nurse Rostering (Martin et al., 2013) using real-world data. This flexibility is achieved by means of an ontology which enables the agents to represent these problems with the same internal structure.

This is an interesting and little researched topic that warrants further investigation such as: extending the ontology to apply the framework to new problems; adding more heuristics and metaheuristics and improving the pattern finding protocol.

Finally, this framework will be published as an open source project so that other metaheuristics 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 website: http://simonpmartin.github.io/macs/.

Acknowledgement

The Engineering and Physical Sciences Research Council (EPSRC) supported this work through the following project: Dynamic Adaptive Automated Software Engineering (DAASE) EP/J017515/1.

Appendix A. Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.ejor.2016.02.045.

References

Altınel, İ. . K., & Öncan, T. (2005). A new enhancement of the Clarke and Wright savings heuristic for the capacitated vehicle routing problem. *Journal of the Operational Research Society, 56*(8), 954–961.

Augerat, P., Belenguer, J., Benavent, E., Corberán, A., Naddef, D., & Rinaldi, G. (1995). Computational results with a branch and cut code for the capacitated vehicle routing problem. *Rapport de recherche-IMAG*.

Aydin, M., & Fogarty, T. (2004a). A distributed evolutionary simulated annealing algorithm for combinatorial optimisation problems. *Journal of Heuristics, 10*(3), 269–292.

Aydin, M., & Fogarty, T. (2004b). Teams of autonomous agents for job-shop scheduling problems: An experimental study. *Journal of Intelligent Manufacturing, 15*(4), 455–462.

Barbucha, D. (2014). A cooperative population learning algorithm for vehicle routing problem with time windows. *Neurocomputing, 146*, 210–229.

Bellifemine, F. L., Caire, G., & Greenwood, D. (2007). *Developing multi-agent systems with JADE*. Wiley.

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

Burke, E., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., et al. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society, 64*(12), 1695–1724.

Clarke, G., & Wright, J. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research, 12*(4), 568–581.

Clearwater, S. H., Hogg, T., & Huberman, B. A. (1992). Cooperative problem solving. *Computation: The micro and the macro view*, 33–70.

Crainic, T., & Toulouse, M. (2008). Explicit and emergent cooperation schemes for search algorithms. In *Proceedings of the international conference on learning and intelligent optimization* (pp. 95–109).

Dantzig, G., & Ramser, J. (1959). The truck dispatching problem. *Management Science, 6*, 80–91.

El Hachemi, N., Crainic, T. G., Lahrichi, N., Rei, W., & Vidal, T. (2014). *Solution integration in combinatorial optimization with applications to cooperative search and rich vehicle routing*.

FIPA (2009). FIPA iterated contract net interaction protocol specification. http://www.fipa.org/specs/fipa00030/index.html.

Garey, M., & Johnson, D. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. Bell Telephone Laboratories Inc.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research, 1*(2), 117–129.

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

Hogg, T., & Williams, C. P. (1993). Solving the really hard problems with cooperative search. In *Proceedings of the national conference on artificial intelligence* (pp. 231–236).

Hutter, F., Babic, D., Hoos, H., & Hu, A. J. (2007). Boosting verification by automatic tuning of decision procedures. In *Proceedings of the formal methods in computer aided design, FMCAD'07* (pp. 27–34). IEEE.

Juan, A., Faulin, J., Grasman, S. E., Rabe, M., & Figueira, G. (2015). A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. *Operations Research Perspectives, 2*, 62–72.

Juan, A., Ruíz, R., Lourenço, H. R., Mateo, M., & Ionescu, D. (2010a). A simulation-based approach for solving the flowshop problem. In *Proceedings of the winter simulation conference* (pp. 3384–3395).

Juan, A. A., Faulin, J., Jorba, J., Caceres, J., & Marquès, J. M. (2013). Using parallel & distributed computing for real-time solving of vehicle routing problems with stochastic demands. *Annals of Operations Research, 207*(1), 43–65.

Juan, A. A., Faulín, J., Jorba, J., Riera, D., Masip, D., & Barrios, B. (2011). On the use of monte carlo simulation, cache and splitting techniques to improve the clarke and wright savings heuristics. *Journal of the Operational Research Society, 62*(6), 1085–1097.

Juan, A. A., Faulin, J., Ruiz, R., Barrios, B., & Caballé, S. (2010b). The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem. *Applied Soft Computing, 10*(1), 215–224.

Juan, A. A., Lourenço, H. R., Mateo, M., Luo, R., & Castella, Q. (2014). Using iterated local search for solving the flow-shop problem: Parallelization, parametrization, and randomization issues. *International Transactions in Operational Research, 21*(1), 103–126.

Kouider, A., & Bouzouia, B. (2012). Multi-agent job shop scheduling system based on co-operative approach of idle time minimisation. *International Journal of Production Research, 50*(2), 409–424.

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2011). The IRACE package, iterated race for automatic algorithm configuration. *Tech. Rep. TR/IRIDIA/2011-004*. Belgium: IRIDIA, Université Libre de Bruxelles.

Malek, R. (2010). An agent-based hyper-heuristic approach to combinatorial optimization problems. In *Proceedings of 2010 IEEE international conference on Intelligent computing and intelligent systems (ICIS): Vol. 3* (pp. 428–434).

Martin, S., Ouelhadj, D., Smet, P., Vanden Berghe, G., & Özcan, E. (2013). Cooperative search for fair nurse rosters. *Expert Systems with Applications, 40*(16), 6674–6683.

Meignan, D., Creput, J., & Koukam, A. (2008). A coalition-based metaheuristic for the vehicle routing problem. In *Proceedings of IEEE congress on evolutionary computation, 2008, CEC 2008. (IEEE world congress on computational intelligence)* (pp. 1176–1182). IEEE.

Meignan, D., Koukam, A., & Créput, J. C. (2010). Coalition-based metaheuristic: A self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 1–21.

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

Moore, D. S., & McCabe, G. P. (1989). *Introduction to the practice of statistics*. WH Freeman/Times Books/Henry Holt & Co.

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

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

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

Ries, J., & Beullens, P. (2015). A semi-automated design of instance-based fuzzy parameter tuning for metaheuristics based on decision tree induction. *Journal of the Operational Research Society, 66*(5), 782–793.

Ross, P. (2014). Hyper-heuristics. In *Search methodologies* (pp. 611–638). Springer.

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

Talbi, E. G., & Bachelet, V. (2006). Cosearch: A parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms, 5*(1), 5–22.

Toulouse, M., Thulasiraman, K., & Glover, F. (1999). Multi-level cooperative search: A new paradigm for combinatorial optimization and an application to graph partitioning. *Euro-Par'99 parallel processing*, 533–542.

Vallada, E., & Ruiz, R. (2009). Cooperative metaheuristics for the permutation flowshop scheduling problem. *European Journal of Operational Research, 193*(2), 365–376.

Xie, X. F., & Liu, J. (2009). Multiagent optimization system for solving the traveling salesman problem (TSP). *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 39*(2), 489–502.

Zobolas, G., Tarantilis, C. D., & Ioannou, G. (2009). Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research, 36*(4), 1249–1267.