A Cooperative Hyper-heuristic Search Framework

Djamila Ouelhadj[1] and Sanja Petrovic

Automated Scheduling, Optimisation and Planning Research Group

School of Computer Science,

University of Nottingham, NG8 1BB, UK.

Email: {dxs, sxp}@cs.nott.ac.uk.

Abstract In this paper, we aim to investigate the role of cooperation between low level heuristics within a hyper-heuristic framework. Since different low level heuristics have different strengths and weaknesses, we believe that cooperation can allow the strengths of one low level heuristic to compensate for the weaknesses of another. We propose an agent-based cooperative hyper-heuristic framework composed of a population of heuristic agents and a cooperative hyper-heuristic agent. The heuristic agents perform a local search through the same solution space starting from the same or different initial solution, and using different low level heuristics. The heuristic agents cooperate synchronously or asynchronously through the cooperative hyper-heuristic agent by exchanging the solutions of the low level heuristics. The cooperative hyper-heuristic agent makes use of a pool of the solutions of the low level heuristics for the overall selection of the low level heuristics and the exchange of solutions. Computational experiments carried out on a set of permutation flow shop benchmark instances illustrated the superior performance of the cooperative hyper-heuristic framework over sequential hyper-heuristics. Also, the comparative study of synchronous and asynchronous cooperative hyper-heuristics showed that asynchronous cooperative hyper-heuristics outperformed the synchronous ones.

Keywords *hyper-heuristics, cooperative search, distributed problem solving, permutation flow shop scheduling*

---

[1] Corresponding author's new contact details are: University of Portsmouth, Department of Mathematics, Lion Gate Building, PO1 3HF. Email: djamila.ouelhadj@port.ac.uk.

1 Introduction

1.1 Hyper-heuristics

Hyper-heuristics have emerged as a new search methodology that is motivated by the goal of increasing the level of generality of meta-heuristics. One of the motivations for studying hyper-heuristics is to build general domain-independent search methodologies that are capable of performing well-enough, soon enough, and cheap-enough across a wide range of optimisation problems (Burke et al. 2003a; Ross 2005).

The term hyper-heuristic has only been introduced recently in 2000 (Soubeiga 2003). However, the origin of the idea can be traced back to the early 1960s (Fisher and Thompson 1963). The term hyper-heuristic has been defined to describe a high level methodology for choosing or generating heuristics to solve combinatorial optimisation problems (Soubeiga 2003; Burke et al. 2003a, 2009; Ross 2005). The hyper-heuristic operates at a higher level of abstraction without knowledge of the domain in which it operates. It only has access to a set of low level heuristics. The low level heuristics are simple local search operators or domain dependent heuristics. Unlike meta-heuristics which search in a space of solutions, hyper-heuristics search in a space of low level heuristics.

Hyper-heuristic approaches so far can be classified into two main categories (Soubeiga 2003; Burke et al. 2009). In the first class, *heuristics to choose heuristics*, the hyper-heuristic uses a set of known domain dependent low level heuristics. In the second class, *heuristics to generate heuristics*, the hyper-heuristic evolves new low level heuristics by making use of the components of the existing ones. These two main classes can be further categorised according to whether the hyper-heuristic controls construction or perturbation low level heuristics. A hyper-heuristic that controls construction low level heuristics builds a solution incrementally. It starts with an empty solution, and then selects the most suitable construction heuristics to gradually build a complete solution. A hyper-heuristic that controls perturbation low level heuristics starts with a complete initial solution and iteratively selects the appropriate perturbation heuristics to improve the current solution. Most of the hyper-heuristics that control perturbation low level heuristics proposed in the literature perform a single point search, processing a single

solution at each iteration. They are generally composed of two phases (Ozcan et al. 2008): heuristic selection strategy and solution acceptance criteria. The heuristic selection strategy selects the most appropriate low level heuristic to apply at each iteration. The solution generated by the selected low level heuristic is either accepted or rejected based on a solution acceptance criterion. An additional classification of hyper-heuristics considers the source providing feedback during the learning process, which can be either on-line or off-line. In on-line learning hyper-heuristics, the learning takes place while the algorithm is solving an instance of a problem. An example of on-line learning is the use of meta-heuristics as high-level search strategies over a search space of heuristics. In off-line learning hyper-heuristics, the idea is to gather knowledge in the form of rules or programs, from a set of training instances, which would be used to solve unseen instances. Examples of off-line learning are: learning classifier systems, case-based reasoning, and genetic programming. The rest of the paper focuses on the first category, heuristics to choose heuristics, and in particular on hyper-heuristics that control perturbation low level heuristics.

Early research work on hyper-heuristics emphasized the development of advanced selection strategies. Fisher and Thompson (1963) are the first researchers to use the idea of a hyper-heuristic for the job shop scheduling problem. They proposed random hyper-heuristics and a hyper-heuristic based on probabilistic weighting to guide the selection of the low level heuristics. Soubeiga (2003) proposed random, greedy, and choice function hyper-heuristics. Each selection strategy uses two acceptance criteria which are AM (All Moves) where all moves are accepted, and IO (Improving Only) where only improving moves are accepted. The random hyper-heuristic selects randomly the next low level heuristic to apply at each decision point of the search. The greedy hyper-heuristic applies all the low level heuristics, and selects the best improving low level heuristic. The choice function hyper-heuristic uses reinforcement learning to guide the choice of the low level heuristics. The hyper-heuristic uses a choice function to select a low level heuristic at each decision point. The choice function accumulates historical information about the recent performance of each low level heuristic. Nareyek (2003) used a non-stationary reinforcement learning method to select the low level heuristics. In this approach, each low level heuristic is assigned a weight which can increase or

decrease according to the low level heuristic performance. Various reward and punishment schemes were considered when selecting a low level heuristic. More advanced learning mechanisms have also been investigated within a hyper-heuristic framework including meta-heuristics, case based reasoning, learning classifier systems, etc. Cowling at al. (2003) proposed a tabu search based hyper-heuristic to solve the personnel scheduling problem. In the tabu search hyper-heuristic, a tabu list was incorporated to prevent the selection of low level heuristics with poor performance for a certain number of iterations. At each iteration, the hyper-heuristic selects greedily the best low level heuristic. If such a heuristic leads to an improved objective function value it is always selected and released from the tabu list if there; a non-improving heuristic is chosen only if it is not in the tabu list and immediately becomes tabu after its application. Burke et al. (2003b) proposed a tabu search hyper-heuristic with reinforcement learning to solve the nurse rostering and university course timetabling problems. They used reinforcement learning for the selection of low level heuristics. Ross et al. (2002) used a learning classifier system to learn, for a given stage of bin-packing problems, which heuristics were more useful than others. Cowling et al. (2002) proposed a GA based hyper-heuristic to solve a trainer scheduling problem. The GA chromosome represents an ordering of the low level heuristics to be applied to the current state. Burke et al. (2007) used a tabu search algorithm to select well-known graph colouring heuristics in a hyper-heuristic framework to solve exam and course timetabling problems. Burke et al. (2006) developed a case-based hyper-heuristic for timetabling problems which selects low level heuristics based on their performance in previous similar situations. Burke et al. (2005) proposed an ant based hyper-heuristic to solve the presentation scheduling problem.

Recently, research has been carried out to improve the solution acceptance criteria in a hyper-heuristic framework. Bai et al. (2008), Soubeiga (2003), and Dowsland et al. (2007) introduced a simulated annealing acceptance criterion into the hyper-heuristic framework. Ayob and Kendall (2003) proposed a Monte Carlo based hyper-heuristic which accepts improving and non improving low level heuristics using a probabilistic framework. Kendall and Mohamad (2004) used the great deluge algorithm as the acceptance criterion. Landa Silva and Obit (2009) proposed a non-linear great deluge

acceptance criterion. Ozcan et al. (2008) in their experimental study on hyper-heuristics, combined some of the above different selection strategies and acceptance criteria and compared their performance on benchmark exam timetabling problems. The experimental results showed that no combination of heuristic selection strategies and acceptance criteria can dominate others. Different combinations might perform better in different domains.

Literature review on hyper-heuristics has mainly concentrated on the development of sequential hyper-heuristics where a single low level heuristic is selected at a time and applied to a single working solution. A hyper-heuristic framework is inherently distributed and very suitable to distributed problem solving as it consists of a set of low level heuristics directed by a high level hyper-heuristic. Distributing the hyper-heuristic framework opens up the possibility of having parallel execution of multiple low level heuristics that can cooperate by sharing many different working solutions to initiate the low level heuristics. Indeed, since different low level heuristics have different strengths and weaknesses, it makes sense to see whether they can cooperate in some way so that the strengths of one low level heuristic compensates for the weaknesses of another (Burke et al. 2003a).

1.2 Cooperative parallel search

The last ten to fifteen years have witnessed a continuously stronger stream of important developments in parallel cooperative optimisation most of which targeted meta-heuristics including tabu search, genetic algorithms, and simulated annealing (Crainic et al. 1997; Crainic and Toulouse 2003; Alba 2005; Aydin 2007). The main goals of parallel meta-heuristics are to reduce the overall computation time required to solve a problem instance, and enhance the robustness of the search by performing a broader search of the solution space with a comparable computation effort or, at least, for the same computation time. In some cases, this may even lead to a more efficient search scheme capable of finding better solutions than the corresponding sequential search approach. Robustness can be obtained by the use of different combinations of strategies and parameter settings at each processor, leading to high quality solutions for different instances of the same problem, without need for parameter tuning.

Crainic and Toulouse (2003) proposed a classification scheme for parallel meta-heuristic strategies. They grouped parallel meta-heuristic strategies into three categories: low level parallelisation (type 1), parallelisation by domain decomposition (type 2), and multi-thread strategies (type 3). In type 1, parallelism is usually found within an iteration where moves are evaluated in parallel. The meta-heuristic is implemented on a master processor. At each iteration, slave processors evaluate in parallel the possible moves in the neighbourhood of the current solution. A computationally expensive part of a heuristic is parallelised with the sole purpose of deriving run time speedups. Type 2 or decomposition strategy consists in partitioning the search space of the problem into several sets and running the meta-heuristic on each subset, thus accelerating the global search. The resulting partial explorations are combined by a master process to obtain a feasible solution. In type 3 or multi-thread strategies, threads search through the same solution space, starting from possibly the same or different initial solutions and using possibly different meta-heuristics or the same ones with different parameter settings. The threads may communicate during the search or only at the end to identify the best overall solution. The latter are known as independent search methods, while the former are often called cooperative search methods. Interest in cooperative search has risen considerably among researchers due to its success to provide novel ways to combine several search algorithms. Clearwater et al. (1992), Hogg and Williams (1993), Talbi and Bachelet (2006), and Aydin (2007) pointed out that multi-agent systems (Ferber, 1999) propose natural ways to efficiently implement cooperative search. Blum and Roli (2003), Clearwater et al. (1922), Hogg and Williams (1993), Toulouse et al. (1999), and Crainic and Toulouse (2008) described cooperative search as a search performed by agents that exchange information about states, models, entire sub-problems, solutions or other search space characteristics. Aydin (2007) described meta-heuristic agents as multi-agent systems that have the potential of carrying out concurrent search within search spaces. The agents implement the same or different search algorithms, and cooperate by exchanging useful information on the search. The key challenge in cooperative search is the design of a cooperation mechanism and the determination of useful information to exchange between the agents. Inter-agent cooperation may be performed synchronously or asynchronously, and directly or indirectly (Crainic and

Toulouse 2008). The island model in the evolutionary paradigm uses generally direct cooperation. The population is divided into subsets, each assigned to a processor, and a genetic algorithm runs on each. An individual population and a genetic algorithm form an island. Each island may communicate with any of other islands. Most developments on cooperative search to deal with complete solutions outside the evolutionary paradigm are based on indirect inter-agent cooperation using a memory called pool (central memory, warehouse, blackboard, etc.). The pool stores the best solutions found so far by the agents. The search agents send to the pool good solutions, and recuperate solutions from the pool to diversify the search. This model has been successfully used to solve a number of difficult combinatorial optimisation problems, such as multi-commodity location with balancing requirements (Crainic et al. 1995a, b), capacitated network design (Crainic and Gendreau 2002), vehicle routing problem (Le Bouthillier and Crainic 2005), quadratic assignment (James et al. 2009), labour constraint scheduling (Cavalcante et al. 2001), etc. Furthermore, comparative studies (Crainic et al. 1997) conducted in the literature showed the effectiveness and the superiority of synchronous and asynchronous cooperative search in terms of the solution quality and computation time compared with the sequential ones. Studies have also shown that asynchronous cooperative search performed best.

To the best of our knowledge, there is scarce research work on cooperative search in a hyper-heuristic framework. Gaw et al. (2004) introduced two variants of asynchronous distributed choice function hyper-heuristics for University timetabling. In the first one, the overall control of the search is managed by a single hyper-heuristic running on one processor and many processors executing each only one low level heuristic. The processors may execute the same low level heuristic on different parts of the timetable. The second variation consists of multiple choice function hyper-heuristics, each with their own subordinate processors. Recently, Ouelhadj and Petrovic (2008, 2009) proposed a generic cooperative hyper-heuristic framework where the hyper-heuristic involves the cooperation of the low level heuristics in the selection process. The cooperative hyper-heuristic framework involves a single hyper-heuristic and the cooperation of a population of low level heuristics. The framework is composed of a cooperative hyper-heuristic agent and a number of low level heuristic agents. The

cooperative hyper-heuristic agent is in charge of the selection of the low level heuristic to apply at a decision point of the search space. The low level heuristic agents search through the same solution space and cooperate synchronously or asynchronously through the cooperative hyper-heuristic agent by exchanging the solutions of the low level heuristics.

In this paper, we aim at furthering research into the investigation of the role of cooperation between low level heuristics within a hyper-heuristic framework. We propose an agent-based cooperative hyper-heuristic framework to deal with complete solutions based on cooperative meta-heuristic search described above. We also investigate synchronous and asynchronous cooperation between low level heuristics based on the solution pool strategy proposed by Crainic and Toulouse (2008), and propose a variety of cooperative hyper-heuristic approaches.

The rest of the paper is organised as follows. Section 2 presents the cooperative hyper-heuristic framework. Section 3 describes the synchronous and asynchronous cooperative hyper-heuristic search mechanisms. Section 4 discusses the implementation and the experimental results obtained on a set of permutation flow shop benchmark instances. Conclusions are presented in section 5.

2 The generic cooperative hyper-heuristic framework

The cooperative hyper-heuristic framework is an agent-based system composed of a population of independent heuristic agents which cooperate through a cooperative hyper-heuristic agent, as illustrated in Fig. 1. The heuristic agents and the cooperative hyper-heuristic agent are described below.

2.1 Heuristic agents

Each low level heuristic is assigned to a heuristic agent. The heuristic agents perform a local search on complete solutions to improve their local solutions using each a different low level heuristic and starting from the same or different initial solutions. They search in the same space of solutions. The heuristic agents cooperate through the cooperative hyper-heuristic agent by exchanging their local best solutions in order to combine the efforts of several independent low level heuristics, and to improve the quality of the

solutions that each of them would be able to find by itself working on a stand alone basis. The local best solutions are the complete best solutions found by the heuristic agents at each search cycle. A search cycle comprises all computations performed by a heuristic agent until it communicates its new local best solution.

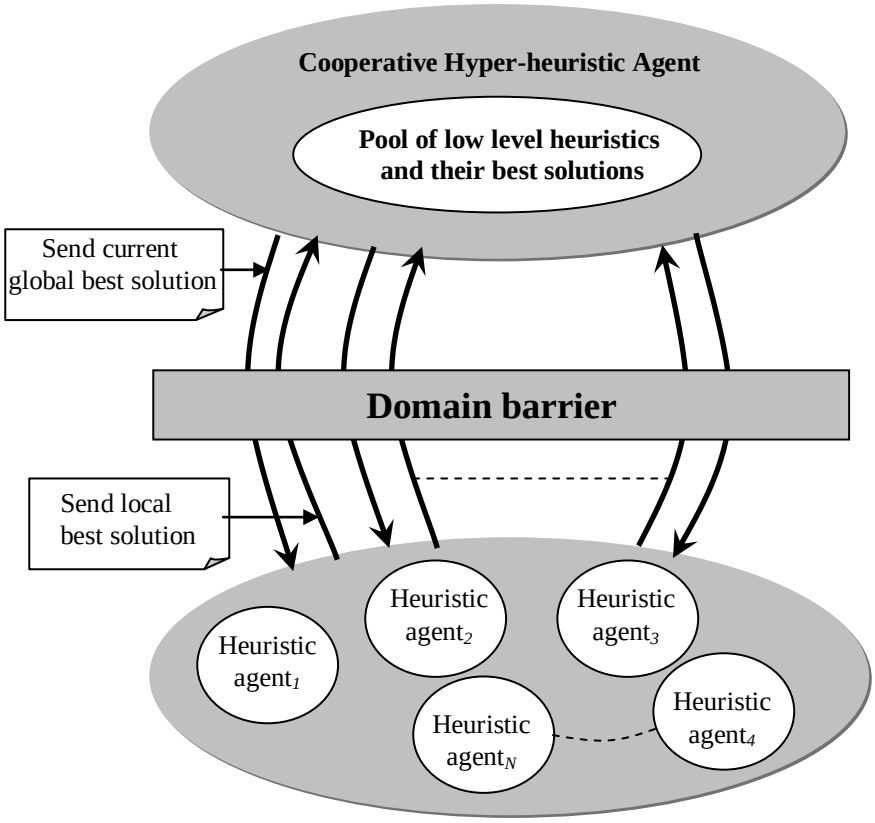

Fig. 1 A cooperative hyper-heuristic search framework

2.2 Cooperative hyper-heuristic agent

The cooperative hyper-heuristic agent manages the cooperation between the heuristic agents, the overall selection of the low level heuristics, and the acceptance of their solutions. It operates at a higher level of abstraction without knowledge of the domain under which it operates. It only has access to a set of low level heuristics. The domain barrier in Fig. 1 shows that there is no domain knowledge exchanged between the heuristic agents and the cooperative hyper-heuristic agent.

2.2.1 Cooperation mechanism

Cooperation between the low level heuristics is based on the solution pool strategy proposed by Crainic and Toulouse (2008). The cooperative hyper-heuristic agent maintains a pool which stores the low level heuristics and their local best solutions sent by the heuristic agents. Each entry in the pool contains the following: the name of the low level heuristic, its local best solution, and the objective function value. The heuristic agents send their local best solutions to the pool and the cooperative hyper-heuristic agent provides them with solutions from the pool to diversify the search. The cooperative hyper-heuristic agent keeps also the current global best solution and the overall global best solution. The current global best solution is the current best solution found by all the heuristic agents within a search cycle. It is updated at the end of each search cycle. The global best solution is the overall best solution found by the heuristic agents.

2.2.2 Selection of low level heuristics and solution acceptance criteria

At each search cycle, the heuristic agents perform a local search and send their local best solutions to the cooperative hyper-heuristic agent. Then, the cooperative hyper-heuristic agent uses a greedy selection strategy for the selection of the low level heuristic with the best solution in the pool. The selected low level heuristic either improves the current global best solution or not if there are no improving low level heuristics. The cooperative hyper-heuristic agent then decides whether to accept or not the selected solution to be sent to the heuristic agents to diversify the search using the following solution acceptance criteria:

- **Improving Only (IO) criterion**: The cooperative hyper-heuristic agent accepts the solutions of the low level heuristics only if they improve the current global best solution.
- **All Moves (AM) criterion**: The cooperative hyper-heuristic agent accepts the solutions of the low level heuristics even if they do not improve the current global best solution.

- **Tabu Search (TS) criterion**: The cooperative hyper-heuristic agent accepts all the solutions of low level heuristics even if they do not improve the current global best solution. However, a non improving low level heuristic is made tabu in a tabu list for a certain number of iterations, set to 7, to prevent its solutions from being accepted too soon (Cowling et al. 2003). The solution of a tabu low level heuristic that improves the current global best solution is accepted and the low level heuristic is released from the tabu list. The solution of a non improving low level heuristic is accepted only if the low level heuristic is not in the tabu list, but the low level heuristic immediately becomes tabu.

- **Simulated Annealing (SA) criterion**: The cooperative hyper-heuristic agent accepts the solutions of the low level heuristics whether they improve the current global best solution or not. Solutions of non improving low level heuristics are accepted with a probability $exp\ (\Delta/T)$, where Δ is the change in the objective function value and T the temperature. The acceptance probability is controlled using a temperature parameter which is gradually decreased using a cooling schedule. We use the geometric cooling schedule, where the temperature is typically decreased by a factor $\mu = 0.85$ (Soubeiga 2003). The initial temperature is set to 50% of the value of the initial solution. The temperature is high at the beginning of the search leading to a high acceptance probability to allow for a wider exploration of the search space, and gradually decreases as the search progresses to allow for intensification.

- **Great Deluge (GD) criterion**: The cooperative hyper-heuristic agent accepts all the solutions of the low level heuristics whether they improve the current global best solution or not. Solutions of non improving low level heuristics which produce objective function values less than a certain level are accepted. The acceptance level decreases over time by a fixed decay rate $\alpha = (f(s_o)-LB) / num\text{-}iter$, where $f(s_o)$ is the objective function value of the initial solution, LB is the lower bound of the objective function (best solution found in the literature), $num\text{-}iter$ is the maximum number of iterations allocated for the search (Kendall and Mohamed 2004).

3 Cooperative search

The heuristic agents perform a local search on a complete solution using the assigned low level heuristics. They cooperate through the cooperative hyper-heuristic agent by exchanging the local best solutions of the low level heuristics stored in the pool. We propose two cooperative search variants: synchronous and asynchronous. The two variants are detailed below.

3.1 Synchronous cooperative search

In synchronous cooperative search, the heuristic agents communicate with the cooperative hyper-heuristic agent to exchange their local best solutions at the end of each cycle. A cycle consists of a predetermined number of iterations. The cooperative hyper-heuristic agent has to wait for all the local best solutions of the heuristic agents before proceeding to the selection and solution acceptance steps. The synchronous cooperative search is detailed in the following steps:

- The cooperative hyper-heuristic agent manages the pool of low level heuristics, their best solutions, the current global best solution, and the overall global best solution.
- In each search cycle, the cooperative hyper-heuristic agent broadcasts a request to the heuristic agents to perform a local search starting from the same solution for a predetermined number of iterations.
- The heuristic agents search through the same solution space using each a different low level heuristic, and starting from the same solution for a predetermined number of iterations or until no further improvement is possible. Then, each of them sends its local best solution to the cooperative hyper-heuristic agent.
- Upon receiving the local best solutions from all the heuristic agents, the cooperative hyper-heuristic agent inserts in the pool each local best solution, its low level heuristic, and its objective function value. The objective function of each local best solution could be better, worse, or of the same value as that of the

current global best solution. Note that at the end of each cycle the pool is reinitiated with the current local best solutions generated by the heuristic agents

- The cooperative hyper-heuristic agent selects from the pool the best of all the low level heuristics and its solution.
- If the selected best solution is better than the current global best solution, then the cooperative hyper-heuristic agent updates the current global best solution with the selected best solution and requests the heuristic agents to start a new search cycle starting from the new current global best solution. The heuristic agents carry out a diversification phase starting from the new current global best solution.
- If the selected best solution is worse than the current global best solution, then the cooperative hyper-heuristic agent decides whether to accept it or not using the following acceptance criteria: IO, AM, TS, SA, and GD. If the selected best solution is accepted then the cooperative hyper-heuristic agent updates the current global best solution and requests the heuristic agents to start a new search cycle starting from the new current global best solution. The heuristic agents carry out a diversification phase starting from the new current global best solution. Otherwise, the cooperative hyper-heuristic agent checks whether to accept the next best solution in the pool.
- The search stops after a maximum number of cycles, or a number of cycles without improvement, or there are no solutions in the pool to send to the heuristic agents.

We propose five variants of synchronous cooperative hyper-heuristic search which use the greedy selection strategy and the IO, AM, TS, SA, and GD acceptance criteria. The five variants are the following: Synchronous Cooperative IO (SC-IO), Synchronous Cooperative AM (SC-AM), Synchronous Cooperative TS (SC-TS), Synchronous Cooperative SA (SC-SA), and Synchronous Cooperative GD (SC-GD). The heuristic agent pseudo-code is the same for all the synchronous cooperative hyper-heuristic approaches and is given in Fig. 2. In Fig. 3, we present the generic cooperative hyper-heuristic agent pseudo-code since all the synchronous cooperative hyper-heuristic

approaches differ only in the solution acceptance criteria. Note that in the pseudo-codes, we refer to the Heuristic Agent as HA and Cooperative Hyper-heuristic Agent as CHA. We also consider the minimisation of the objective function *f*.

Receive request search (CHA, HA, $S_{current\text{-}global\text{-}best}$, iters-cycle);
1. Initialisation: Set local best solution, $S_{local\text{-}best}$ to $S_{current\text{-}global\text{-}best}$ at the first cycle;
2. Local search and intensification
If $S_{current\text{-}global\text{-}best} = S_{local\text{-}best}$ then perform a local search to improve $S_{local\text{-}best}$ using a low level heuristic;
3. Diversification
If $S_{current\text{-}global\text{-}best} != S_{local\text{-}best}$ then $S_{local\text{-}best} = S_{current\text{-}global\text{-}best}$; go to 2;
4. Termination: Stop if iters-cycle or until no further improvement is possible;
send (HA, CHA, $S_{local\text{-}best}$); otherwise go to 2;

Fig. 2 Heuristic agent pseudo-code in synchronous cooperative search

1. Initialisation
Set initial solution; set pool of low level heuristics and their solutions to nil; set current global best solution, $S_{current\text{-}global\text{-}best}$ to initial solution; set overall best solution, $S_{global\text{-}best}$ to initial solution; set tabu-list, tabu- list length, , level, T,α, μ; set maximum number of cycles, max-num-cycles and number of iterations of a cycle, iters-cycle; set number of cycles without improvement, num-cycles-no-imp;
2. Selection of low level heuristics and solution acceptance criteria
Broadcast request search (CHA, HAs, $S_{current\text{-}global\text{-}best}$, iters-cycle);
Receive local best solutions (HAs, CHA, $S_{local\text{-}best}$);
Insert the low level heuristics and their $S_{local\text{-}best}$ in pool;
Select $S_{local\text{-}best}$ of the best improving or non-improving low level heuristic LHH from pool;
If f ($S_{local\text{-}best}$) < $S_{current\text{-}global\text{-}best}$ then
$S_{current\text{-}global\text{-}best} = S_{global\text{-}best} = S_{local\text{-}best}$;
End if
If f ($S_{local\text{-}best}$) >= $S_{current\text{-}global\text{-}best}$ then
While $S_{local\text{-}best}$ not accepted
Do
Check whether to accept or not $S_{local\text{-}best}$ using IO, AM, TS, SA, GD acceptance criteria;
If $S_{local\text{-}best}$ accepted then $S_{current\text{-}global\text{-}best} = S_{local\text{-}best}$; $S_{local\text{-}best}$ accepted;
else select $S_{local\text{-}best}$ of the next best LLH in pool;
End if
End while
End if
3. Termination: Stop if max-num-cycles or num-cycles-no-imp
or all $S_{local\text{-}best}$ in pool not accepted; otherwise go to 2;
output $S_{global\text{-}best}$;

Fig. 3 Generic cooperative hyper-heuristic agent pseudo-code in synchronous cooperative search

3.2 Asynchronous cooperative search

In asynchronous cooperative search, the communications with the cooperative hyper-heuristic agent are initiated exclusively by the heuristic agents based on their local solutions and timings. Whenever a heuristic agent improves its current local best solution, it sends the new local best solution to the cooperative hyper-heuristic agent. Similarly, when a heuristic agent cannot improve its current local best solution after a certain number of iterations, it requests a solution from the cooperative hyper-heuristic agent. The straightforward advantage of asynchronous cooperative search is that the heuristic agents are not idle compared to synchronous cooperative search. The asynchronous cooperative search is undertaken as follows.

- The cooperative hyper-heuristic agent manages the pool of low level heuristics, their best solutions, the current global best solution, and the overall global best solution.
- The cooperative hyper-heuristic agent broadcasts a request to the heuristic agents to perform a local search starting from the same solution for a maximum number of iterations.
- The heuristic agents search through the same solution space using each a different low level heuristic, and starting from the same initial solution.
- Each time a low level heuristic improves its local solution, the heuristic agent sends the new local best solution to the cooperative hyper-heuristic agent.
- If the new solution is better than the current global best solution, the cooperative hyper-heuristic agent inserts it in the pool and updates the current global best solution.
- If the new solution is worse than the current global best solution, the cooperative hyper-heuristic agent checks whether to accept it or not using IO, AM, TS, SA, and GD acceptance criterion. If the new solution is accepted then the cooperative hyper-heuristic agent inserts it in the pool and updates the current global best solution. Otherwise, it sends the current global best solution to the heuristic agent to diversify the search. Note that the cooperative hyper-heuristic agent sends to

the heuristic agents only the solutions that they have not received previously. Otherwise, the next best solution from the pool is sent to the heuristic agents.

- Whenever a low level heuristic cannot improve its local best solution, the heuristic agent requests a solution from the cooperative hyper-heuristic agent to diversify the search.
- The search stops when all the heuristic agents stop the search after a maximum number of iterations or there are no solutions in the pool to send to the heuristic agents.

Similarly to synchronous cooperative search, we propose five variants of asynchronous cooperative hyper-heuristic search which use the different acceptance criteria described in section 3.1. The asynchronous cooperative hyper-heuristic approaches are the following: ASynchronous Cooperative IO (ASC-IO), ASynchronous Cooperative AM (ASC-AM), ASynchronous Cooperative TS (ASC-TS), ASynchronous Cooperative SA Hyper-Heuristic (ASC-SA), and ASynchronous Cooperative GD (ASC-GD). The heuristic agent pseudo-code described in Fig. 4 is the same to all the asynchronous cooperative hyper-heuristic approaches. Fig. 5 describes the generic pseudo-code of the cooperative hyper-heuristic agent.

Receive request search (CHA, HA, initial solution, max-iters);
1. Initialisation: Set local best solution $S_{local\text{-}best}$ to initial solution; set max-iters; set number of iterations without improvement, num-no-imp;
2. Local search and intensification
Perform a local search to improve $S_{local\text{-}best}$ using a low level heuristic LLH;
If LLH improves $S_{local\text{-}best}$ then
send (HA, CHA, $S_{local\text{-}best}$); receive (CHA, HA, acceptance status, $S_{solution}$);
If CHA accepts $S_{local\text{-}best}$ then go to 4 else go to 3;
End if
3. Diversification
If LLH does not improve $S_{local\text{-}best}$ for num-no-imp then
Request solution from CHA; receive (CHA, HA, $S_{solution}$); $S_{local\text{-}best} = S_{solution}$; go to 4;
Endif
If $S_{local\text{-}best}$ is not accepted then $S_{local\text{-}best} = S_{solution}$; go to 4;
4. Termination: Stop if max-iters or CHA is unable to return $S_{solution}$; otherwise go to 2;

Fig. 4 Heuristic agent pseudo-code in asynchronous cooperative search

1. Initialisation: Set initial solution; set pool of low level heuristics and their local-best solutions to nil; set current global best solution, $S_{current\text{-}global\text{-}best}$ to initial solution; set overall best solution, $S_{global\text{-}best}$ to initial solution; set maximum number of iterations, max-iters; set tabu-list, tabu-list length, level, T, α, μ;

Broadcast request search (CHA, HAs, initial solution, max-iters);

2. Selection of low level heuristics and solution acceptance criterion

Whenever a new $S_{local\text{-}best}$ is sent from HA

Do

If f ($S_{local\text{-}best}$) < f ($S_{current\text{-}global\text{-}best}$) then

If pool not full then insert $S_{local\text{-}best}$ and its low level heuristic LHH in pool;

else $S_{local\text{-}best}$ replaces the worst solution in pool;

End if

$S_{current\text{-}global\text{-}best}$ = $S_{global\text{-}best}$ = $S_{local\text{-}best}$;

Send an acceptance confirmation to HA to continue its search from $S_{local\text{-}best}$;

Endif

If f ($S_{local\text{-}best}$) >= f ($S_{current\text{-}global\text{-}best}$) then

Check whether to accept or not $S_{local\text{-}best}$ using IO, AM, TS, SA, GD acceptance criteria;

If $S_{local\text{-}best}$ accepted then

If pool not full then insert $S_{local\text{-}best}$ and LLH in pool;

else $S_{local\text{-}best}$ replaces the worst solution in pool;

End if

$S_{current\text{-}global\text{-}best}$ = $S_{local\text{-}best;}$

Send an acceptance confirmation to HA to continue the search from $S_{local\text{-}best}$;

End if

If $S_{local\text{-}best}$ not accepted then

If HA has not received before $S_{current\text{-}global\text{-}best}$ then send (CHA, HA, $S_{current\text{-}global\text{-}best}$);

else $S_{solution}$ = next best solution in pool HA has not received before; send (CHA, HA, $S_{solution}$);

End if

End if

End do

Whenever a HA requests a solution

Do

If HA has not received before $S_{current\text{-}global\text{-}best}$ then send (CHA, HA, $S_{current\text{-}global\text{-}best}$);

else $S_{solution}$ = next best solution in pool HA has not received before; send (CHA, HA, $S_{solution}$);

End if

End do

3. Termination: Stop when all the HAs stop; otherwise go to 2; output $S_{global\text{-}best}$;

Fig. 5 Generic cooperative hyper-heuristic agent pseudo-code in asynchronous cooperative search

4 Simulation prototype and computational experiments

A simulation prototype was developed to implement the generic cooperative hyper-heuristic framework. The prototype was implemented in C# using multi-threads. The cooperative hyper-heuristic agent and heuristic agents are implemented as multi-thread Windows applications. The communication between agents is implemented using Microsoft Message Queuing (MSMQ). The communication language developed for inter-agent communication is based on speech act theory which uses performatives to express the actions of the agents (Ferber 1999). The powerful expression of the performatives provides the agents with a natural form of conversation. The communication language is expressed in XML (eXtensible Mark-up Language).

The prototype provides a powerful interactive graphical interface for the user to select the low level heuristics and the cooperative hyper-heuristic approaches, and to follow the steps of the cooperative search. The prototype is very modular and flexible to integrate new or remove low level heuristics and cooperative hyper-heuristic approaches. In terms of reusability, the prototype is generic enough to be reused in different application domains.

To evaluate the performance of the cooperative hyper-heuristic framework, we conducted several experiments on permutation flow shop scheduling benchmark instances.

4.1 Case study: Permutation flow shop scheduling

Permutation flow shop scheduling is defined as the problem of sequencing jobs to be processed on machines (Pinedo 1995) in which the jobs must be processed in the same sequence on all the machines. Each job can be processed on only one machine at a time, and each machine can process only one job at a time. Each job is processed only once on each machine. Operations are not preemptable and set up times of operations are independent of the sequences and therefore can be included in the processing times.

Solving the permutation flow shop scheduling problem consists in finding an optimal schedule of processing n jobs ($j = 1, ..., n$) on m machines ($i = 1, \ldots, m$). A job consists of m operations and the i^{th} operation of each job must be processed on machine i. A job

can start processing on machine *i* if it is completed on machine *i-1* and if machine *i* is free. Each operation of job *j* on machine *i* is assigned a processing time $p_{j,i}$ time unit.

The objective we have considered is the minimisation of makespan C_{max} (the completion time of the last job on the last machine). The completion time $C_{j,i}$ of job *j* on machine *i* is calculated using the following formulae:

$$C_{1,1} = p_{1,1} \quad (1)$$

$$C_{j,i} = C_{j,i-1} + p_{j,i} \quad i = 2 \ldots m \quad (2)$$

$$C_{j,1} = C_{j-1,1} + p_{j,1} \quad j = 2 \ldots n \quad (3)$$

$$C_{j,i} = \max(C_{j-1,i}, C_{j,i-1}) + p_{j,i} \quad i = 2 \ldots m, \; j = 2 \ldots n \quad (4)$$

$$C_{\max} = C_{n,m} \quad (5)$$

4.2 Low level heuristics

The low level heuristics considered are the most common ones used to solve the permutation flow shop scheduling problem. They are as follows:

- **Swap low level heuristic**: It exchanges the positions of two jobs in the sequence of scheduled jobs.
- **Insertion low level heuristic**: It moves a job from its current position to a new position in the sequence of scheduled jobs.
- **Inversion low level heuristic**: It reverses the positions of jobs in a randomly chosen sub-sequence of jobs from the sequence of scheduled jobs.
- **Permutation low level heuristic**: It selects randomly a sub-sequence of jobs from the sequence of scheduled jobs, and then performs a random permutation of the jobs within the sub-sequence.

4.3 Experimental results

In the experiments, we implemented the following four heuristic agents: swap agent, insertion agent, inversion agent, and permutation agent. We considered the flow shop

benchmark problems for makespan given by Taillard (Taillard 1993, 2007) and presented in the Operational Research library (OR). The benchmark set is composed of 120 different instances of different sizes. Each instance specifies the processing times of n jobs on m machines, where n = {20, 50, 100, 200} and m = {5, 10, 20}. For each combination $n \times m$, the benchmark contains 10 instances. In total there are 12 combinations and these are: 20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×10, 200×20, and 500×20.

The heuristic agents use the same initial solution generated using the *NEH* heuristic for permutation flow shop scheduling (Nawaz et al. 1983). This heuristic first orders the jobs by decreasing total required processing time on the machines. Then, it schedules the first two jobs in order to minimise the partial makespan as if there were only these two jobs. The rest of the jobs are added one at a time at the positions which minimise the makespan.

In the first set of experiments, we compared the performance of the synchronous and asynchronous cooperative hyper-heuristic approaches with their sequential counterparts reported in the literature to solve personal scheduling, timetabling, self space allocation, and channel assignment problems. In order to make a fair comparison, we re-implemented the sequential hyper-heuristics with the suggested parameter values described in section 2.2.2. The sequential hyper-heuristics we implemented are the following:

- Greedy IO (GR-IO) and Greedy AM (GR-AM) (Soubeiga 2003) both implement the greedy selection strategy and the IO and AM acceptance criteria.
- TS hyper-heuristic (TS-HH) (Cowling et al. 2003) implements the greedy selection strategy and the TS acceptance criterion.
- SA hyper-heuristic (SA-HH) (Soubeiga 2003) and GD hyper-heuristic (Kendall and Mohamad 2004) both implement simple random selection strategy and the SA and GD acceptance criteria.

All the methods run on an Intel Pentium M 1500 MHz with 512 Mbytes of main memory. The performance measure used is the average percentage increase over the Lower Bounds (LB) of makespan produced by Taillard (1993, 2007). LB is the

makespan of the given optimum solution for each instance or the lowest known upper bound if the optimum for that instance is still unknown.

For each problem size, the average percentage increase over the LB of C_{max} for sequential, synchronous and asynchronous cooperative hyper-heuristic approaches was computed over 10 instances. The increase is computed as follows:

$$\%\ makespan\ \ increase\ over\ LB = \frac{C_{\max} - LB}{LB} \times 100$$

The stopping condition of the sequential hyper-heuristics is a maximum number of 50000 iterations. An iteration involves a makespan evaluation. In the synchronous cooperative hyper-heuristic framework, the maximum number of iterations of each heuristic agent is 12500 (50000/4, 4 is the number of heuristic agents). The number of iterations within a cycle is 100. Thus, the stopping condition of the cooperative hyper-heuristic agent is a maximum number of cycles set to 125. In the asynchronous cooperative hyper-heuristic framework, the stopping condition of the heuristic agents is a maximum number of iterations set to 12500 (50000/4). The size of the pool of best solutions is set to the number of low level heuristics. Each sequential and cooperative hyper-heuristic approach is run 10 independent times.

The results in Table 1 show the average percentage makespan increase over LB of the 10 instances for each problem size, and the last row in the table presents the average percentage makespan increase over LB of all the instances. The results in Table 1 clearly show that on average the synchronous and asynchronous cooperative hyper-heuristics outperformed sequential hyper-heuristics. Both synchronous and asynchronous cooperative hyper-heuristics yield minor average increase in makespan over the LB. Asynchronous cooperative hyper-heuristics yield smaller average increase in makespan (in bold) than synchronous ones. The best results (in bold) were achieved by asynchronous cooperative hyper-heuristics. In addition, both synchronous and asynchronous cooperative hyper-heuristics that use TS, SA, and GD acceptance criteria yield a smaller average increase in makespan than the ones that use IO and AM acceptance criteria. We can also observe that the average increase in makespan over LB of the hyper-heuristics that use the AM acceptance criterion are slightly smaller than the ones that use the IO acceptance criterion. It seems that IO acceptance criterion has a

greater tendency than the AM acceptance criterion to get stuck early in a local optimum. However, we clearly need statistical analysis to make conclusions. ASC-TS, ASC-SA, and ASC-GD performed best even with basic parameter settings, and ASC-SA is the best of all. Our initial experiments suggest that the acceptance of non improving low level heuristics using a meta-heuristic based learning mechanism yield better results because the solutions returned to the requesting heuristic agents are more diverse which avoids the search to get stuck in local optima. Further studies will investigate this issue.

To validate the statistical significance of the observed differences, we performed an analysis of variance (ANOVA) (Montgomery 2000) with Tukey intervals. Fig. 6 shows the means plot at 95% confidence level for the different methods. The statistical analysis results indicated that there are statistically significant differences between the sequential, synchronous and asynchronous cooperative hyper-heuristics with a *p*-value close to zero. The synchronous cooperative hyper-heuristics improved the sequential hyper-heuristics by an average improvement in the range of (9%, 13%). However, better improvements (in bold) were achieved by asynchronous cooperative hyper-heuristics in the range of (24%, 31%). The asynchronous cooperative hyper-heuristics improved the synchronous ones by an average improvement in the range of (14%, 18%). This is mainly due to the predetermined synchronisation points that make the synchronous cooperative hyper-heuristics less reactive to the progress of the search. Furthermore, the heuristic agents are often idle waiting for the end of the cycle to exchange their local best solutions to complete the search. Consequently, introducing asynchronism improved the algorithmic performance. With regard to the significance of the difference between the effectiveness of the IO and AM acceptance criteria, the statistical analysis showed that the small observed differences in the average increase in makespan over LB are not significant. The hyper-heuristics that use the AM acceptance criterion are better than the ones that use the IO acceptance criterion by an average improvement of 0.29% for sequential hyper-heuristics, 1.97% for synchronous cooperative hyper-heuristics, and 1.74% for asynchronous cooperative hyper-heuristics.

We can conclude that asynchronous cooperative hyper-heuristics show a very good performance, improving significantly the results of sequential hyper-heuristics in the literature. Considering that the asynchronous cooperative hyper-heuristic approaches

achieved very minor average increase in makespan over the LB, we may conclude that the results are very promising.

In the second set of experiments, we compared the performance of both the synchronous and asynchronous cooperative hyper-heuristic approaches to the most well performing state of the art sequential meta-heuristics developed for permutation flow shop scheduling, including tabu search, simulated annealing, genetic algorithms, iterated local search, ant algorithms, and hybrid techniques. Again, the measure used in the comparison is the percentage makespan increase over LB. The meta-heuristics considered are (Ruiz and Maroto 2005): simulated annealing (SAOP), tabu search (Spirit), Iterated local search (ILS), GA algorithms (GAReev, GAChen, GAPAC), and finally hybrid GA with local search (GAMIT). For the average percentage makespan increase over LB of the sequential meta-heuristics, we used the results published by Ruiz and Maroto (2005).The stopping criterion used in the paper is 50000 makespan evaluations for all the meta-heuristics. Note that the computational environment of the cooperative hyper-heuristics is not the same as the meta-heuristic one. However, using the number of makespan evaluations criterion makes the comparison fair even if we use different computing platforms. Table 2 shows the average percentage makespan increase over LB of each method for each problem size, and the last row presents the average of all the problems. As expected, the results show that cooperative hyper-heuristic approaches did not outperform the best meta-heuristics (in bold) in the literature. However, they are quite competitive with some of them. Hyper-heuristic are designed to work well over a variety of problem domains without parameter tuning and they are not expected to outperform meta-heuristics which are tailored for specific problems.

To validate our conclusions, we performed an analysis of variance (ANOVA). Fig. 7 gives the means and confidence intervals for the various hyper-heuristics and meta-heuristics. As can be seen many of the differences are statistically significant. SOAP, GAChen, GAReev, and ILS are statistically better than synchronous and asynchronous cooperative hyper-heuristics. However, Spirit and GAMIT are quite similar to synchronous and asynchronous cooperative hyper-heuristics. Furthermore, synchronous and asynchronous cooperative hyper-heuristics outperformed CAPAC. We believe that the average percentage increase in makespan of asynchronous cooperative hyper-

heuristics in the range of (6.89%, 8.17%) makes asynchronous cooperative search a very compelling research direction in hyper-heuristics.

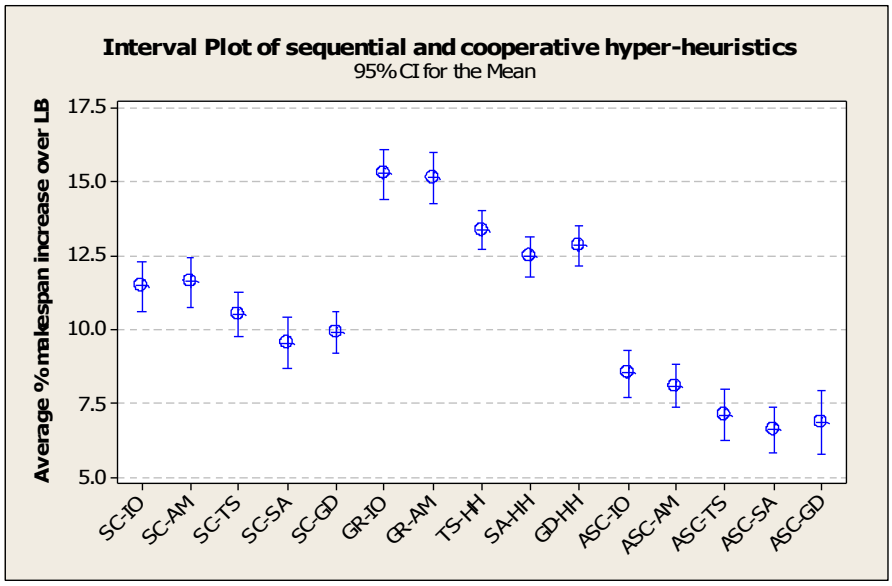

Fig. 6 Means plot and Tukey intervals at 95% confidence level of sequential, synchronous, and asynchronous cooperative hyper-heuristics

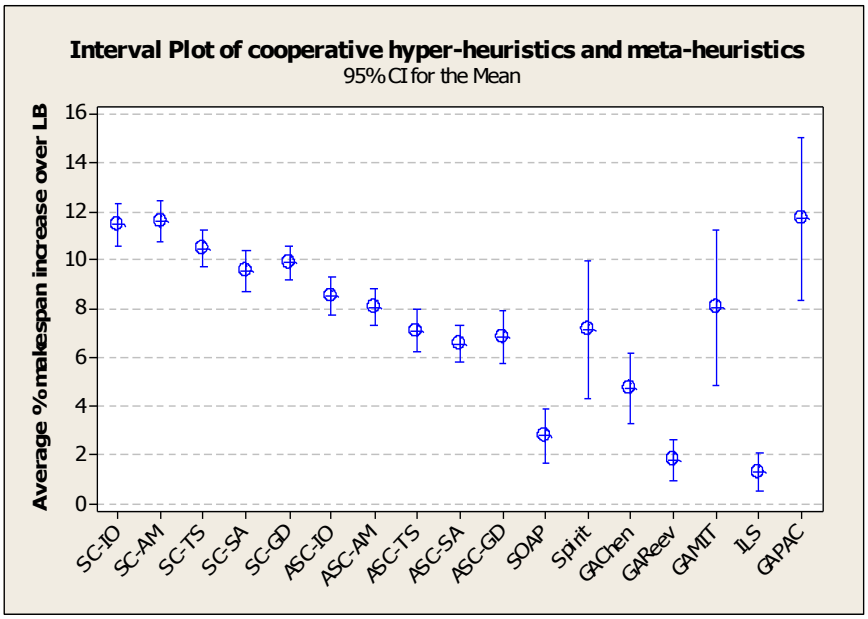

Fig. 7 Means plot and Tukey intervals at 95% confidence level of cooperative hyper-heuristics and meta-heuristics

Table 1 Average percentage makespan increase over LB of sequential and cooperative hyper-heuristics

Instance	Sequential hyper-heuristics					Synchronous cooperative hyper-heuristics					Asynchronous cooperative hyper-heuristics				
	GR-IO	GR-AM	TS-HH	SA-HH	GD-HH	SC-IO	SC-AM	SC-TS	SC-SA	SC-GD	ASC-IO	ASC-AM	ASC-TS	ASC-SA	ASC-GD
20x5	8.28	8.68	8.64	7.23	7.64	6.42	6.34	6.34	6.44	6.54	6.23	6.12	5.87	5.98	**5.65**
20x10	15.04	15.12	13.90	13.89	13.94	14.45	15.97	14.29	14.03	14.15	13.27	13.71	12.94	**12.64**	12.75
20x20	24.78	23.98	23.01	23.98	23.06	24.01	23.61	23.82	22.94	22.88	22.25	21.94	21.9	**19.34**	19.44
50x5	4.33	3.12	3.08	3.16	3.12	3.87	2.56	2.63	2.51	2.74	3.97	3.34	2.97	2.97	**2.18**
50x10	8.16	8.11	7.73	7.50	7.55	7.18	7.14	7.07	6.98	7.03	6.08	6.11	5.21	**4.23**	4.56
50x20	14.46	14.72	15.08	13.67	13.75	13.32	13.71	12.71	12.66	12.61	12.84	12.82	11.24	**10.11**	11.02
100x5	6.17	6.91	5.70	5.23	5.89	4.78	5.38	5.05	5.01	5.06	4.59	4.34	4.42	**4.31**	4.37
100x10	7.28	7.56	6.95	5.23	5.79	6.69	4.97	3.66	3.04	2.98	3.62	3.59	**2.68**	2.78	2.79
100x20	10.15	11.80	9.56	9.67	9.66	10.23	9.54	9.51	8.89	8.06	7.89	7.88	8.12	**7.22**	7.34
200x10	8.62	7.46	7.33	6.35	6.54	6.32	6.66	6.6	5.07	5.58	6.77	5.65	3.98	**3.72**	3.84
200x20	7.69	7.23	7.12	7.04	7.01	6.87	6.59	6.2	6.65	6.19	5.98	6.28	6.25	**6.14**	6.24
500x20	6.67	6.60	6.12	5.98	6.01	7.57	7.11	5.51	4.1	4.72	4.57	4.56	3.88	3.89	**3.78**
Average	10.14	10.11	9.52	9.08	9.16	9.31	9.13	8.62	8.19	8.21	**8.17**	**8.03**	**7.46**	**6.89**	**6.99**

Table 2 Average percentage makespan increase over LB of cooperative hyper-heuristics and the best solutions known for meta-heuristics

	Synchronous cooperative hyper-heuristics					Asynchronous cooperative hyper-heuristics					Meta-heuristics						
Instance	SC-IO	SC-AM	SC-TS	SC-SA	SC-GD	ASC-IO	ASC-AM	ASC-TS	ASC-SA	ASC-GD	SOAP	Spirit	GAChen	GAReev	GAMIT	ILS	GAPAC
20x5	6.42	6.34	6.34	6.44	6.54	6.23	6.12	5.87	5.98	5.65	1.39	5.22	3.82	0.7	4.21	0.24	8.98
20x10	14.45	15.97	14.29	14.03	14.15	13.27	13.71	12.94	12.64	12.75	2.66	5.86	4.89	1.92	5.4	0.77	13.61
20x20	24.01	23.61	23.82	22.94	22.88	22.25	21.94	21.9	19.34	19.44	2.31	4.58	4.17	1.53	4.53	0.85	11.03
50x5	3.87	2.56	2.63	2.51	2.74	3.97	3.34	2.97	2.97	2.18	0.69	2.03	2.09	0.26	3.11	0.12	6.5
50x10	7.18	7.14	7.07	6.98	7.03	6.08	6.11	5.21	4.23	4.56	4.25	5.88	6.6	2.58	8.38	2.01	16.41
50x20	13.32	13.71	12.71	12.66	12.61	12.84	12.82	11.24	10.11	11.02	5.13	7.21	8.03	3.76	10.65	3.29	18.56
100x5	4.78	5.38	5.05	5.01	5.06	4.59	4.34	4.42	4.31	4.37	0.4	1.06	1.32	0.18	5.41	0.11	5.32
100x10	6.69	4.97	3.66	3.04	2.98	3.62	3.59	2.68	2.78	2.79	1.88	5.07	3.75	1.08	12.05	0.66	12.34
100x20	10.23	9.54	9.51	8.89	8.06	7.89	7.88	8.12	7.22	7.34	5.21	10.15	7.94	3.94	18.24	3.17	18.25
200x10	6.32	6.66	6.6	5.07	5.58	6.77	5.65	3.98	3.72	3.84	1.56	9.03	2.7	0.82	7.52	0.49	9.75
200x20	6.87	6.59	6.2	6.65	6.19	5.98	6.28	6.25	6.14	6.24	4.83	16.17	7.07	3.33	15.35	2.74	17.06
500x20	7.57	7.11	5.51	4.1	4.72	4.57	4.56	3.88	3.89	3.78	3.4	13.57	4.61	1.83	2.17	1.29	2.61
Average	9.31	9.13	8.62	8.19	8.21	8.17	8.03	7.46	6.89	6.99	**2.81**	7.15	**4.75**	**1.83**	8.92	**1.31**	12.53

5 Conclusions and future work

In this paper, we have proposed a generic cooperative agent-based hyper-heuristic framework to investigate the role of cooperation between low level heuristics within a hyper-heuristic. The proposed framework is composed of a cooperative hyper-heuristic agent and a population of heuristic agents. The heuristic agents cooperate synchronously or asynchronously through the cooperative hyper-heuristic agent by exchanging the best solutions of the low level heuristics. The cooperative framework is generic and flexible enough to be used to solve a wide range of problem domains. The simulation prototype implemented provides a graphical interface which is very interactive and user friendly.

We have investigated a variety of acceptance criteria, including AM, IO, TS, SA, and GD. We have also proposed a variety of synchronous and asynchronous cooperative hyper-heuristic approaches.

The comparative study has illustrated the effectiveness and superiority of the synchronous and asynchronous cooperative hyper-heuristics over their sequential counterparts. The results have also showed that asynchronous cooperative hyper-heuristic approaches outperformed the synchronous ones. Furthermore, the asynchronous cooperative hyper-heuristic approaches which use meta-heuristics based on-line learning performed best. Although as expected, the cooperative hyper-heuristic approaches did not outperform the state of the art meta-heuristics, we believe that minor makespan deviation from the LB makes cooperative hyper-heuristic approaches very promising. The aim of hyper-heuristics is to build general domain-independent search methodologies that are capable of performing well across a wide range of optimisation problems.

Asynchronous cooperation offers interesting perspectives for future research in cooperative hyper-heuristics. Future research will be devoted to issues related to the cooperation mechanism, in particular nature of the information stored in the pool and the size of the pool, and other cooperation mechanisms. The selection strategy used in the paper is greedy. Current research work is investigating a multi-agent reinforcement learning approach for the selection of low level heuristics. Also, additional experiments will be conducted on other real world scheduling problems.

In order to increase the level of generality of the developed framework, we are currently extending the framework to investigate the role of cooperation between multiple hyper-heuristics to combine the performance of single hyper-heuristics. The heuristic agents implement hyper-heuristics instead of heuristics and the framework implements multiple hyper-heuristics that cooperate asynchronously through the cooperative hyper-heuristic agent to exchange their solutions.

Acknowledgement

We would like to thank the Engineering and Physical Sciences Research Council (EPSRC) in the UK for supporting this research (grant reference EP/D061571/1).

References

Alba, E.: Parallel meta-heuristics: a new class of algorithms. John Wiley & Sons Publishers (2005)

Aydin, M.E.: Meta-heuristic agent teams for job shop scheduling Problems, Lecture Notes in Computer Science, vol. 4659, pp. 185-194 (2007)

Ayob, M., Kendall, G. A.: A Monte Carlo hyper-heuristic to optimise component placement sequencing for multi-head placement machine. In: Proceedings of the International Conference on Intelligent Technologies, Chiang Mai, Thailand, pp.132-141 (2003)

Bai, R., Burke E.K., Kendall G.: Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. Journal of Operational Research Society, 59 (10), 187-197 (2008)

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

Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: an emerging direction in modern search technology. In: Glover, F., Kochenberger, G. (eds.) Handbook of Meta-heuristics, pp. 457-474. Kluwer Academic Publishers (2003a)

Burke, E.K., Kendall, G., Soubeiga, E.: A tabu search hyper-heuristic for timetabling and rostering. Journal of Heuristic, 9 (6), 451-470 (2003b)

Burke, E.K., Kendall, G., Landa Silva, D., O'Brien, R., Soubeiga, E.: An ant algorithm hyper-heuristic for the project presentation scheduling problem. In: Proceedings of the IEEE Congress on Evolutionary Computation, Edinburgh, Scotland, pp. 2263-2270 (2005)

Burke, E.K., Petrovic, S., Qu, R.: Case based heuristic selection for timetabling problems. Journal of Scheduling, 115-132 (2006)

Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyper-heuristic for educational timetabling problems. European Journal of Operational Research, 176 (1), 177-192 (2007)

Burke, E.K, Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: A survey of hyper-heuristics. School of Computer Science, University of Nottingham, Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747 (2009)

Cavalcante, C.C.B., Cavalcante, V.C., Ribeiro, C.C., de Souza, C.C.: Parallel cooperative approaches for the labor constrained scheduling problem. In: Ribeiro, C.C., Hansen, P. (eds.) Essays and Surveys in Meta-heuristics, pp. 201–225. Kluwer Academic Publishers (2001)

Clearwater, S.H., Huberman, B.A., and Hogg, T.: Cooperative problem solving. In: Huberman, B. (eds.) Computation: The Micro and the Macro View, pp. 33-70 (1992)

Cowling, P., Kendall, G., Han, L.: An investigation of a hyper-heuristic genetic algorithm applied to a trainer scheduling problem. In: Proceedings of the IEEE Congress on Evolutionary Computation, Hawaii, pp.1185-1190 (2002)

Cowling, P., Chakhlevitch, K.: Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In: Proceedings of the IEEE Congress on Evolutionary Computation, p. 1214–1221 (2003)

Crainic, T.G., Toulouse, M., Gendreau, M.: Parallel asynchronous tabu search for multi-commodity location allocation with balancing requirements. Annals of Operations Research, 63, 277-299 (1995a)

Crainic, T.G., Toulouse, M., Gendreau, M.: Synchronous tabu search parallelization strategies for multi-commodity location allocation with balancing requirements. OR Spectrum, 17 (2-3), 113–123 (1995b)

Crainic, T.G., Toulouse, M., Gendreau, M.: Towards a taxonomy of parallel tabu search algorithms. INFORMS Journal on Computing, 9 (1), 61-72 (1997)

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

Crainic, T.G., Toulouse, M.: Parallel strategies for meta-heuristics. In: Glover, F., Kochenberger, G. (eds.) Handbook in Meta-heuristics, pp. 475-513. Kluwer Academic Publishers, Norwell, MA (2003)

Crainic, G.T. Toulouse, M.: Explicit and emergent cooperation schemes for search algorithms. Learning and Intelligent Optimization (LION II) Conference, Lecture Notes in Computer Science, vol. 5313, pp. 95-109 (2008)

Dowsland, K.A., Soubeiga, E., Burke, E.K.: A simulated annealing hyper-heuristic for determining shipper sizes. European Journal of Operational Research, 179 (3), 759-774 (2007)

Ferber, J.: Multi-agent systems: An introduction to Distributed Artificial Intelligence. Addison-Wesley, London (1999)

Fisher, H., Thompson, G.L.: Probabilistic learning combinations of local job shop scheduling rules. Prentice Hall, New Jersey, pp. 225-251 (1963)

Gaw, A., Rattadilok P., Kwan R.S.K.: Distributed choice function hyper-heuristics for timetabling and scheduling. In: Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, pp. 495-498 (2004)

Hogg, T., Williams, C.P.: Solving the really hard problems with cooperative search. In: Proceedings of AAAI, pp. 213.235 (1993)

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

Kendall, G., Mohamad, M.: Channel assignment in cellular communication using a great deluge hyper-heuristic. In: Proceedings of the 12th IEEE International Conference on Networks, Singapore, pp. 769-773 (2004)

Landa-Silva, D., Obit, E.J.: Great deluge with nonlinear decay rate for solving course timetabling problems. In: Proceedings of the IEEE Conference on Intelligent Systems, p. 8.11-8.18 (2008)

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

Montgomery, D.C.: Design and Analysis of Experiments. John Wiley & Sons, New York (2000)

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

Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning. In: Resende, M.G.C., de Sousa, J.P. (eds.) Meta-heuristics: Computer Decision-Making. Kluwer Academic Publishers, pp. 523-544 (2003)

OR. http://mscmga.ms.ic.ac.uk/jeb/orlib/flowshopinfo.html. Accessed January 2008

Ouelhadj, D., Petrovic, S.: A cooperative distributed hyper-heuristic framework for scheduling. In: Proceedings of the IEEE International Conference on Man, Cybernetics, and Systems, Singapore, pp.1232-1238 (2008)

Ouelhadj, D., Petrovic, S.: Asynchronous cooperative hyper-heuristic search. In: Proceedings of the International Conference on Artificial Intelligence, Las Vegas, pp. 78-84 (2009)

Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. Intelligent Data Analysis, 12 (1), 3-23 (2008)

Pinedo, M.: Scheduling: Theory, Algorithms and Systems. Prentice Hall (1995)

Ross, P., Schulenburg, S., Marin Blazquez, J.G., Hart, E.: Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In: Proceeding of the Genetic and Evolutionary Computation Conference, New York, USA, pp. 942-948 (2002)

Ross, P.: Hyper-Heuristics. In: Burke, E.K., Kendall, G. (eds.) Search methodologies: Introductory Tutorials in Optimization and Decision Support Techniques. Springer, pp. 529-556 (2005)

Ruiz, R., Maroto, C.: A comprehensive review and evaluation of permutation flow shop heuristics. European Journal of Operational Research, 165 (2), 479-49 (2005)

Soubeiga, E.: Development and application of hyper-heuristics to personnel scheduling. Ph.D. Thesis, University of Nottingham, UK (2003)

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

Taillard, E.D.: Summary of best known lower and upper bounds of Taillard's instances. http://ina.eivd.ch/collaborateurs/etd . Accessed January 2008

Talbi, E., Bachelet, V.: COSEARCH: A parallel cooperative metaheuristic. Journal of Mathematical Modelling and Algorithms, 5, 5-22 (2006)

Toulouse, M., Crainic, T.G., and Sanso, S.: An experimental study of the systemic behavior of cooperative search algorithms. In: Voß, S., Martello, S., Osman, I., Roucairol, C. (eds.) Meta-heuristics: advances and trends in local search paradigms for optimization. Kluwer Academic Publishers, pp. 373.392 (1999)