Information Sciences 314 (2015) 225–239

Contents lists available at ScienceDirect

Information Sciences

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

Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems

Nasser R. Sabar [a,*], Graham Kendall [a,b]

[a] ASAP Research Group, The University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih, Selangor, Malaysia
[b] ASAP Research Group, The University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham, UK

ARTICLE INFO

Article history:
Received 3 March 2014
Received in revised form 5 October 2014
Accepted 12 October 2014
Available online 24 October 2014

Keywords:
Hyper-heuristic
Monte Carlo tree search
Timetabling
Personnel scheduling

ABSTRACT

Hyper-heuristics aim to automate the heuristic selection process in order to operate well across different problem instances, or even across different problem domains. A traditional hyper-heuristic framework has two levels, a high level strategy and a set of low level heuristics. The role of the high level strategy is to decide which low level heuristic should be executed at the current decision point. This paper proposes a Monte Carlo tree search hyper-heuristic framework. We model the search space of the low level heuristics as a tree and use Monte Carlo tree search to search through the tree in order to identify the best sequence of low level heuristics to be applied to the current state. To improve the effectiveness of the proposed framework, we couple it with a memory mechanism which contains a population of solutions, utilizing different population updating rules. The generality of the proposed framework is demonstrated using the six domains of the hyper-heuristic competition (CHeSC) test suite (boolean satisfiability (MAX-SAT), one dimensional bin packing, permutation flow shop, personnel scheduling, traveling salesman and vehicle routing with time windows). The results demonstrate that the proposed hyper-heuristic generalizes well over all six domains and obtains competitive, if not better results, when compared to the best known results that have previously been presented in the scientific literature.

1. Introduction

Meta-heuristic methodologies are used to address problems that cannot be solved by exact methods, due to the size/complexity of the problem. These problems are typically $\mathcal{NP}$-hard. Meta-heuristics have reported many success stories and have tackled problems that would remain intractable if these search methodologies were not available. Meta-heuristics are, and remain, an important addition to the tools available to those who tackle optimization problems, although we may have reached saturation point with regard to the number of meta-heuristic algorithms that we need [54].

A feature of meta-heuristics is that they can often only be used for the problem for which they have been developed. To achieve the results reported in the scientific literature often requires extensive parameter tuning. This is often worthwhile for a given problem instance but it does not provide a return on that investment if we wish to utilize the same meta-heuristic algorithm on another domain. Indeed, a meta-heuristic algorithm may not work well on different problem instances drawn from the same domain, without additional parameter tuning.

* Corresponding author.
E-mail addresses: Nasser.Sabar@nottingham.edu.my (N.R. Sabar), Graham.Kendall@nottingham.edu.my (G. Kendall).

http://dx.doi.org/10.1016/j.ins.2014.10.045
0020-0255/

Hyper-heuristics aim to raise the generality of search algorithms by attempting to design algorithms that are able to operate well across instances drawn from the same domain, as well as being able to operate well on different problem domains. The challenges in achieving this are (at least) twofold. Firstly, we need a software framework that is able to cope with different problem domains, without having to redevelop the software for each domain that we tackle. Secondly, we do not want to tune the algorithmic parameters for each instance/problem. That is, we either want to tune the parameters once (the approach we take in this paper) and then use those settings across all the instances/domains that we are tackling, or we want the hyper-heuristic to adapt to the new instance/problem and tune the parameters as the algorithm executes.

As hyper-heuristic research matures, this methodology could become important to the industrial community, especially smaller companies. They often do not have the expertise, or financial resources, to use complex packages, or have bespoke software developed specifically for them. If a hyper-heuristic package is able to work across several problem domains, or even a common problem domain such as staff scheduling, or vehicle routing, it could provide low cost alternatives to software products that are currently available. Companies are often interested in getting a good quality solution to their problem. They may not require, indeed may not recognize, an optimal solution, but prefer software that is easy to use and can adapt to their problem.

Hyper-heuristic research must be carried out in the context of the No Free Lunch theorem [58]. This says that a general algorithm, that is superior to every other algorithm, across all problems, is not possible. However, that does not stop us attempting to develop algorithms that are more general than those currently available.

In this paper, we propose a new hyper-heuristic framework that has some similarities with exiting frameworks in that there is a high level strategy and a set of low level heuristics. One of the main contributions of this paper is the use of Monte Carlo Tree Search (MCTS) as the high level selection strategy, and the use of a Monte Carlo acceptance criterion to decide if a solution should be accepted or not. We are motivated to use MCTS as it has shown recent, significant success in the game of Go [48] and MCTS is now seen as our current best option to create a Go player that is competitive with the best human players.

We test our proposed algorithm across six different domains, demonstrating that the proposed hyper-heuristic is able to generalize well across a diverse set of problems. The results we present show that this has been achieved, as well as being competitive with the state of the art algorithms and producing new best results.

The structure of the paper is as follows. In the next section we discuss related work before presenting our proposed framework in Section 3. In Section 4, we provide details of our experimental setup and report our results in Section 5. In Section 6, we discuss our results before concluding the paper in Section 7.

2. Related work

2.1. Hyper-heuristics related work

The first appearance of the term *hyper-heuristic* in the scientific literature is in a technical report from 1996 [21]. The term is used to refer to a method of combining artificial intelligence methods in automated theorem proving. A paper the following year [22], with the same title as the technical report, did not explicitly use the term *hyper-heuristic* although the paper is obviously based on the technical report. Both of these works were preceded by a technical report [20], which should be noted for completeness.

It was in 2000 that the term first appeared in a peer reviewed paper [16], in the way that it is now commonly used. However, the roots of what we now regard as hyper-heuristics can be traced back to the 1960s [25,18].

Hyper-heuristics aim to raise the level generality at which search methodologies operate. The holy grail is to have one algorithm that works well across a range of different problems. We know that we can never have a single algorithm for any optimization problem [58] but it does not mean that we cannot aim for more general algorithms than are available today.

Early work on hyper-heuristics (for example [16,32,17,15,46,24]) focused on searching the heuristic space by deciding which heuristic to call at a given point in the search. That is, a set of *low level* heuristics are provided and the *high level* hyper-heuristic algorithm chooses which heuristic to apply at each iteration. One of the motivations for this type of hyper-heuristic is that the set of low level heuristics can be replaced and the hyper-heuristic is able to address a different problem, without changing the high level search algorithm. This type of hyper-heuristic is often referred to as *heuristics to choose heuristics* or a *selection hyper-heuristic* [49,11,30,8].

Hyper-heuristics have been applied to many domains, with timetabling receiving particular attention. Early timetabling/hyper-heuristic papers focused on selection hyper-heuristics. Burke et al. [6] investigated the hybridization of two graph coloring heuristics within a selection hyper-heuristic. Their results demonstrated the generality of the methodology. In [32] a new examination timetabling problem was introduced (the largest known problem in Malaysia). The authors used a tabu search based hyper-heuristic to produce better quality solutions than could be produced using a manually created solution. Rattadilok et al. [47] investigated parallel architectures within a hyper-heuristic framework, using a university timetabling problem as an experimental testbed.

Researchers have also investigated how their hyper-heuristic would cope with more than one domain. In [4], two health care timetabling problems (patient admission and nurse rostering) are studied. The paper demonstrates the effectiveness of the hyper-heuristic in addressing two different problems and was able to produce new best known results for patient

admission. Burke et al. [10] showed that a hyper-heuristic was able to generalize over two timetabling problems and a rostering problem, when using a tabu search based hyper-heuristic.

Selection hyper-heuristics are now being used for multi-objective problems. In [35], the low level heuristics are typical multi-objective algorithms (e.g. NSGA II, MOGA) and the high level heuristic has to decide which algorithm should be utilized at a particular time in the overall search.

Soghier and Qu [53] took a slightly different approach. They investigated hybridizing bin packing heuristics to assign examinations to time slots and rooms. This was combined with a graph coloring heuristic to choose which examination to schedule next. The hyper-heuristic, hybridizing graph coloring and bin-packing heuristics, generalized well, performing comparably to the state of the art approaches.

In [2], the hyper-heuristic adopts the familiar two layer framework (the high level hyper-heuristic and the set of low level heuristics) but also utilizes a simulated annealing acceptance criterion, a stochastic heuristic selection strategy and a short-term memory. The approach is tested on two different problems (university course timetabling and bin packing) and, using the same parameter settings, superior solutions are generated when compared to a tabu search hyper-heuristic [10]. Moreover, some of the solutions were superior to meta-heuristic approaches. This paper is typical of new research directions opening up in hyper-heuristic research. Early work focused on selection hyper-heuristics, where the aim was to choose which heuristic to apply. Bai et al. [2] includes features such as a more sophisticated acceptance criterion (i.e. deciding if to accept a solution returned by a low level heuristic).

One of the current research directions in hyper-heuristics is to generate the low-level heuristics, rather than a user having to supply them. Indeed, a criticism that could be leveled at selection hyper-heuristics is that when you replace the set of heuristics you need to implement the heuristics for the new domain. This is not only time consuming but the question also arises as to which heuristics you implement? Do you implement as many as possible, or just a minimal set in the hope that a good quality solution will still be returned?

Several papers (for example [51,28,9,52,50]) have investigated how heuristics can be evolved. This would mean that the set of low level heuristics could be evolved, rather than having to be implemented.

Hyper-heuristics have also been used in classic optimization problems. Kendall and Li [31] utilize a hyper-heuristic for the Competitive Travelling Salesman Problem. Each *agent* adopts a set of heuristics which they use to decide the order in which they will visit each city. The aim is to visit a city before another agent in order to receive the payoff which is only available to the first agent to reach the city.

A recent study [39] considered the generality of selection hyper-heuristics, across three different problem domains, using fourteen different hyper-heuristics. Their findings show that there are significant differences, under different experimental conditions.

Introductions and surveys to hyper-heuristics are available from [49,11,30,8].

2.2. Hyper-heuristics, memetic algorithms and memetic computing

Memetic Algorithms (MAs) are meta-heuristic algorithms that combine population based algorithms with one, or a set of, local search algorithms [44,42]. In Ong et al. [45] hyper-heuristic frameworks have been classified as a sub-class of MAs, where memes represent the low level heuristics and the high level heuristic of a hyper-heuristic framework is a selection strategy that manages the selection of which meme should be called at any given time. Ong et al. [45] classified MAs into three categories where the hyper-heuristics were defined as second generation MAs. In particular, hyper-heuristics were classified as adaptive MAs [45]. Thus, the hyper-heuristic framework proposed in this paper can be considered as an adaptive MA.

As there are many search methodologies drawing their inspiration from cultural diffusion, but which cannot be categorized under MAs definition, the concept of Memetic Computing (MC) was introduced as a broad subject studying search methodologies that are composed of a set of operators, complex structures or interacting memes [29,14,13,43]. MC was defined [41] as "*a broad subject which studies complex and dynamic computing structures composed of interacting modules (memes) whose evolution dynamics is inspired by the diffusion of ideas. Memes are simple strategies whose harmonic coordination allows the solution of various problems*". According to this definition, there is a relation between hyper-heuristic frameworks and MC. In particular, hyper-heuristic frameworks fall under the definition of MC and they have been classified in the first category of MC approaches [41]. In this context, the proposed hyper-heuristic can be seen as an MC approach.

3. Proposed hyper-heuristic framework

The proposed hyper-heuristic framework contains two levels: a high level strategy and the low level heuristics, as shown in Fig. 1. The high level strategy (see Section 3.1) has two components: the heuristic selection mechanism which manages the selection of which low level heuristic is to be applied and the acceptance criterion which decides whether to accept or reject the solution returned from the execution of the chosen low level heuristic.

The low level heuristic (see Section 3.2) involves three different components: a set of problem specific heuristics which generates a new solution by modifying (perturbing) the selected solution, the memory mechanism which contains a population of solutions and the objective function to evaluate the quality of the generated solution.

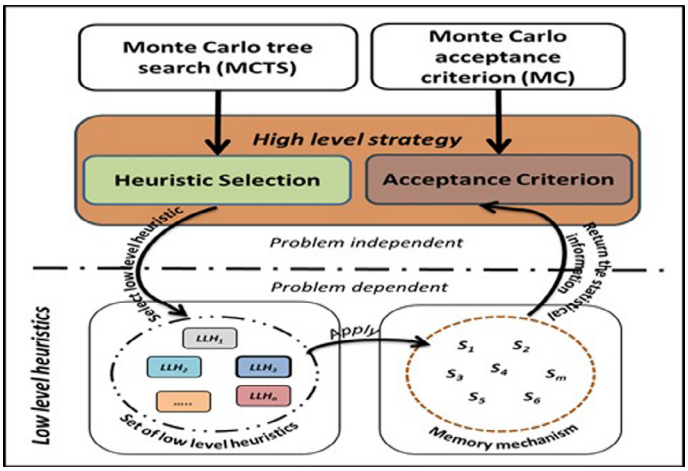

Fig. 1. The proposed hyper-heuristic framework.

In particular, the proposed hyper-heuristic framework starts from an initial complete solution and iteratively explores its neighborhood solutions seeking for a better one by executing the following steps for a certain number of iterations. The detailed steps are as follows:

1. Call the heuristic selection mechanism to select a low level heuristic from a given set.
2. Randomly select a solution from the memory mechanism.
3. Apply the selected low level heuristic to the selected solution and call the objective function to evaluate the quality of the generated solution.
4. If the generated solution is better than the current one, replace it with the current solution. Otherwise, call the acceptance criterion to either accept or reject the generated solution.
5. Update the hyper-heuristic parameters and check the stopping condition.
6. If the stopping condition is satisfied, stop and return the best solution. Otherwise, go to Step 1.

In what follows, we describe the proposed high level strategy together with its components, and the low level heuristics that will be used.

3.1. Proposed high level strategy

Since different problem instances have different search space characteristics, the high level strategy is an important part of a hyper-heuristic framework. A high level strategy should be able to adapt to different problems and/or instances, selecting an appropriate low level heuristic at any given choice point. Designing an intelligent high level strategy that can select an appropriate low level heuristic for the current state is a challenging task, as the search landscape changes over time.

In this paper, we propose a new high level strategy for the proposed hyper-heuristic framework that attempts to work well over different problem domains by utilizing the current state of the search to select the most suitable low level heuristic.

One of the contributions of this paper is the use of Monte Carlo tree search as a high level heuristic selection mechanism. We also use a Monte Carlo acceptance criterion in order to accept or reject the newly generated solution.

3.1.1. Basic Monte Carlo tree search algorithm

The Monte Carlo Tree Search algorithm (MCTS) [5] is a best-first search method that uses random exploration to explore the search space. Based on the results of the previously explored space, MCTS incrementally builds the search tree in memory and further develops the search tree by searching the space of the most promising moves. MCTS has four steps, outlined in Fig. 2, which are executed for a certain number of iterations. These steps are:

1. **Selection:** the selection process starts from the root node and iteratively selects a child node, based on previous information, to develop a tree path that ends at a leaf node. The selection process controls the exploitation–exploration tradeoff. That is, moves that lead to promising results are preferred (exploitation) but, due to the uncertainty in the search space, moves which look less promising should also be explored (exploration). A widely used selection mechanism is the upper confidence bounds (UCB) strategy [5]. The selection process stops once a leaf node is reached.
2. **Expansion:** the expansion step decides whether one, or a few new nodes, should be added to the current child. The simplest expansion strategy is to simply add one new node at each iteration. This strategy will prevent the tree from becoming too large, too quickly, and will reduce memory usage. Once an unexplored node has been added to a current partial tree path the simulation process is then invoked.

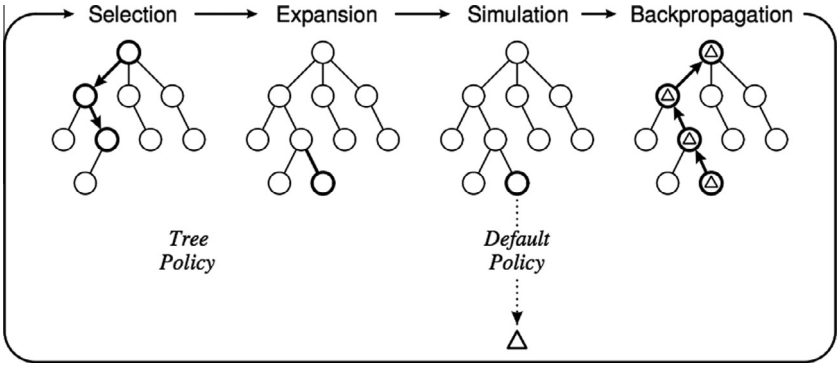

Fig. 2. MCTS steps [5].

3. **Simulation:** the simulation strategy starts from the newly added nodes and executes the generated tree. In particular, moves are selected either randomly or heuristically based on the chosen strategy.
4. **Backpropagation:** backpropagation propagates the results of the simulation strategy over the generated tree path; starting from the leaf node and ending at the root node. In particular, this process updates the node statistic information, for use in later expansions. The nodes will receive a positive reward if the generated tree *won* (we use game playing terminology as MCTS has had recent success in this domain) and gets a negative reward if the generated tree *lost*.

3.1.2. The proposed Monte Carlo tree search hyper-heuristic framework

The search space of low level heuristics, in a hyper-heuristic context, is usually considered as an optimization problem. The main role of the high level strategy is to search such a space, seeking for the appropriate low level heuristic to be utilized for the current problem state. As there is no prior knowledge of which low level heuristic will be the most effective, deciding which one to call is difficult, and requires a fast and efficient decision procedure. Furthermore, since each problem instance has a different landscape, low level heuristic performance varies, not only for different problem instances, but also during different phases of the search. To address this difficulty, in this work, we model the search space of the low level heuristics as a tree and propose a Monte Carlo tree search algorithm to search this tree, seeking for the best sequence of low level heuristics to be applied to a given solution. To this end, each node of the search tree represents a low level heuristic and the generated partial tree is considered as a complete state if the search reaches a leaf node (the last node in the current path based on the tree depth). That is, the search tree represents the low level heuristic search space. Accordingly, the four steps of the basic MCTS are changed to represent the high level strategy of the hyper-heuristic framework as follows:

1. **Selection:** starting from the root node, select a node to be added to the current tree path using the multi-armed bandit (MAB) selection mechanism which uses the upper confidence bound principle. Formally, let k represent the number of available nodes (low level heuristics). Each node is associated with its empirical reward q_{it} (Eq. (2)) (the average reward r_i obtained by the ith node up to time t) and a confidence level n_{it} (the number of times that the ith node has been applied up to time t). MAB selects the best node (low level heuristic) that maximizes the accumulative reward over time (Eq. (1)):

$$max_{i=1\ldots k}\left(q_{it}+c\sqrt{\frac{2log\sum_{j=1}^{k}n_{jt}}{n_{it}}}\right) \tag{1}$$

where c ($c = 12$, see Table 8) is a scaling factor which controls the trade-off between the node that has the best reward (the left term of Eq. (1)) and the node that has been infrequently applied (the right term of Eq. (1)). q_{it} is the empirical reward which is calculated as shown in Eq. (2):

$$q_{it+1}=\left(\frac{n_{it-1}\times q_{it}+r_{it}}{n_{it}}\right) \tag{2}$$

where r_{it} is the accumulative reward value of the ith node up to time t calculated as follows (minimization problems): assume f_1 is the quality of the current solution of a given problem instance and f_2 is the quality of the resultant solution after applying the node (low level heuristic), $r_{it} = ((f_1 - f_2)/f_1) \times 100$. MAB stops once a leaf node is selected. The output of this process is a sequence of nodes that represent a sequence of low level heuristics in the hyper-heuristic framework.
2. **Expansion:** at each iteration, one node is added to the current partial tree path. In this work, to explore a new area in the low level heuristic search space, we randomly select an unexplored node (a low level heuristic that is not included in the current partial tree path) and add it to a current partial tree path.
3. **Simulation:** in the simulation step, the hyper-heuristic framework starts from the leaf of the generated tree and successively applies the nodes of the generated tree to a given solution. Then, the outcome of the generated tree is calculated in terms of the improved gain in the objective function of the solution before and after applying the generated tree.

4. **Backpropagation:** in this step, the improvement obtained by each node is calculated using Eq. (2) and the node statistical information is updated, i.e., update the empirical reward (qi) and the confidence level (ni) of each node.

The proposed high level strategy of the proposed hyper-heuristic framework uses MCTS as a heuristic selection mechanism and the Monte Carlo mechanism (MC) as an acceptance criterion. The main role of MCTS is to search the space of the low level heuristics and generate, at each call, a sequence of low level heuristics to be applied to a given problem solution. MC is used to decide whether to accept or reject the solutions that are returned by the low level heuristics. In MC, improving solutions are always accepted. Worse solutions are accepted with a probability of $R < exp(-\delta)$ [1]. The pseudocode of the proposed high level strategy is presented in Algorithm 1.

Algorithm 1. The proposed high level strategy

```
1  Initialization;
2  Set tree depth, n, tree width, m, and memory size;
3  for i=1 to n do
4  |   for j=1 to m do
5  |   |   tree(i,j)=randomly select one node;
6  |   end
7  end
8  for i=1 to the memory size do
9  |   Randomly generate a solution;
10 |   Add the generated solution to the memory mechanism;
11 |   Calculate the fitness value (f) of the generated solution;
12 end
13 for i=1 to the number of low level heuristics (nodes) do
14 |   Randomly select one solution from the memory;
15 |   Apply the ith low level heuristic (node) on this solution;
16 |   Update the low level heuristic empirical reward (q_i) using eq. 2;
17 |   Update the confidence level (n_i), n_i=n_i+1;
18 end
19 Set maximum iterations, MaxIt, It=0;
20 while It <MaxIt do
21 |   // selection step //;
22 |   current_node=root_node;
23 |   while current_node is not a leaf do
24 |   |   last_node=current_node;
25 |   |   current_node= select node using eq. 1;
26 |   end
27 |   //expansion step //;
28 |   Randomly add a new node that is not included in the current tree path;
29 |   S=Randomly select one solution from the memory;
30 |   // simulation step //;
31 |   for i= last_node to root_node do
32 |   |   S‘=Apply node(i) on the selected solution, S;
33 |   |   if f(S‘) <= f(S) then
34 |   |   |   S=S‘;
35 |   |   else
36 |   |   |   δ= f(S‘)-f(S);
37 |   |   |   p=exp(−δ);
38 |   |   |   r= generates a random number between zero and one;
39 |   |   |   if r <p then
40 |   |   |   |   S=S‘;
41 |   |   |   end
42 |   |   |
43 |   |   end
44 |   end
45 |   // backpropagation step//;
46 |   for i= last_ node to root_ node do
47 |   |   Update the node empirical reward (q_i) using eq. 2;
48 |   |   Update the confidence level (n_i), n_i= n_i+1;
49 |   end
50 |   // update the memory mechanism //;
51 |   Randomly select one of the updating rules to add the new solution, S, to the memory;
52 |   // update the iteration counter //;
53 |   It=It+1;
54 end
55 Return the best solution;
```

The proposed algorithm works as follows: first, set the tree depth (n), tree width (m) and the memory size (line 2). Next, the hyper-heuristic starts from the root node (first node in the tree) and then randomly selects m nodes for the current tree width until the depth of the tree equal is to n (lines 3–7). Then, initialize the memory mechanism (see Section 3.2)

(lines 8–12), calculate the performance of each low level heuristic (node) using Eq. (2) (lines 13–18) and set the maximum iterations counter (*MaxIt*) (line 19). Next, the while-loop is executed (lines 20–54). In each iteration, call the selection step of MCTS to generate a partial tree (lines 22–26) that starts at root node and ends at the leaf node using Eq. (1). Next, call MCTS expansion step to add one node to the current tree (randomly add a new node that is not included in the current partial tree path) (line 28). Then, randomly select one solution, S, from the memory mechanism (line 29). Call the simulation step of MCTS to apply the nodes of the generated tree to the selected solution, starting from the added node (leaf node) and ending at the root (lines 31–44). Each node (low level heuristic) of the generated tree is applied to the current solution, S, to generate a new one, S' (line 32). Next, call the objective function to calculate the quality of S' (line 33). Replace the S with S' if S' is better (line 34). Otherwise, call the Monte Carlo acceptance criterion to decide whether to accept or reject S' (lines 35–43). Next, call MCTS backpropagation to update node statistical information using Eq. (2) (lines 46–49). Randomly select one updating rule to update the memory mechanism (line 51) and update the iteration counter (line 53). If the stopping condition is satisfied stop and return the best solution. Otherwise, start a new iteration.

3.2. The low level heuristics

The low level heuristics of the proposed hyper-heuristic framework involves three different components:

1. A pool of perturbative low level heuristics: in this work, a set of a problem specific heuristics are utilized as low level heuristics. These low level heuristics are different from one problem to another and are used to generate neighborhood solutions by perturbing the current solution. The utilized low level heuristics are restricted to only explore the feasible region of the solution search space, i.e., generate solutions that do not violate any hard constraints. Details of these perturbative heuristics are presented in the problem description section (see Section 4.1).
2. Memory mechanism: to diversify the search process, the proposed hyper-heuristic framework operates on a population of solutions, instead of a single solution, whereas most of the existing hyper-heuristic frameworks deal with only a single solution. The idea is to enhance the effectiveness of the hyper-heuristic framework by using a population of solutions that are scattered over the solution search space. We are motivated by the fact that a single solution is not well suited to cope with large search spaces and heavily constrained problems and thus a population of solutions could possibly do better. In this work, the proposed hyper-heuristic framework uses a memory mechanism that saves a set of solutions. These solutions are generated either randomly or using a heuristic method (see problem description, Section 4.1). At each iteration, the proposed hyper-heuristic framework will randomly pick one solution from the memory, apply the low level heuristics to this solution and update the memory based on the updating rules. We use three different update rules to add a new solution into the memory mechanism. Using different updating criteria enables some worse solutions to be added to the memory, which promotes population diversity and prevents the solutions from being located in the same area. At each iteration, the proposed hyper-heuristic will randomly select one updating rule to update the memory mechanism. That is, all updating rules have equal selection probability. The proposed updating rules are:
   (a) If the new solution is better than the worst one in the memory, it will replace the worst solution in memory.
   (b) The new solution will replace the worst solution in memory regardless of its quality.
   (c) The new solution will be replaced with a solution randomly selected from memory.
3. Objective function: the objective function is problem dependent and is used to evaluate the quality of the generated solution. Different problem domains have different objective functions. The description of the objective functions for each of the problems domains we address is given in Section 4.1.

4. Experimental setup

In this section, we briefly discuss the problem domains that have been used to evaluate the performance of the proposed Monte Carlo tree search hyper-heuristic framework (denoted as MCTS-HH) against the best available hyper-heuristic frameworks, followed by the parameter settings of MCTS-HH.

4.1. Problem description

The proposed hyper-heuristic framework is investigated using the Hyper-heuristics Flexible Framework software (HyFlex) [7,38]. HyFlex was used during the cross-domain heuristic search challenge competition (CHeSC) to facilitate the comparisons between various hyper-heuristic frameworks using the same set of problem domains and also to help researchers in developing a general hyper-heuristic framework for a diverse set of problem domains [7]. HyFlex is a Java framework that has six problem domains (boolean satisfiability (MAX-SAT), one dimensional bin packing, permutation flow shop, personnel scheduling, traveling salesman and vehicle routing), with five instances for each problem domains (30 instances in total). In HyFlex, all problem specific aspects such as the initial solution generation method and the set of perturbative low level heuristics are provided. The researcher only needs to integrate their high level strategy within HyFlex and test it over the provided problem domains. In the following, we briefly describe the HyFlex problem domains and the set of perturbative low level heuristics that are provided.

4.1.1. Problem domain I: Boolean satisfiability

Boolean Satisfiability is a well-known optimization problem [23,12]. The problem can be defined as determining the assignment of truth values to the variables of a given Boolean formula, in order to make the formula true. HyFlex integrates MAX-SAT, which is an extension of Boolean Satisfiability, that seeks to maximize the number of true clauses of a given Boolean formula. The objective is to minimize the number of unsatisfied clauses. Table 1 presents the instance characteristics of the MAX-SAT that have been used in this work. The set of initial solutions are obtained by assigning either a true or false value to each variable and the quality of the solutions (objective function) is measured based on how many 'broken' clauses there are in a given formula i.e., those which evaluate to false. An example of a formula (Boolean equation) is given in Eq. (3) [23]:

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \tag{3}$$

A possible solution to this formula that satisfies the three clauses is when $x_1 = true,\ x_2 = false, x_3 = true$ and $x_4 = true$. See [23] for more details.

4.1.2. Problem domain II: One dimensional bin packing

The one dimensional bin packing problem is the process of packing a set of items, each with a given weight, into a finite number of bins of fixed capacity [34]. The goal of the optimization process is to minimize the number of bins in such a way that each item is only assigned to one bin only and the total weight of items in each bin should be less than or equal to the bin capacity. Table 2 summarizes the characteristics of the problem instances. The set of initial solutions are generated using the first fit heuristic as follows: first, generate a random sequence of items and then pack them one by one into the first bin which they will fit. The quality of the solutions (objective function) is calculated using Eq. (4).

$$quality = 1 - \frac{1}{n}\sum_{i=1}^{n}\left(\frac{f_i}{C}\right)^2 \tag{4}$$

where n is the number of bins, f_i represents the fullness of bin i which is the sum of the weights of all the items in bin i, and C is the bin capacity. See [27] for more details.

4.1.3. Problem domain III: Permutation flow shop

In the permutation flow shop problem, given a set of jobs, each one requires processing on a set of machines, with each operation taking a given time. The task is to find the sequence of jobs on each machine, minimizing the completion time of the last job to exit the shop [55]. The generated sequence should respect the following constraints: a machine can only process one job at a time; jobs can be processed by only one machine at a time, the job ordering process should be respected and machines are not allowed to remain idle when a job is ready for processing. The instances that are used in this work are summarized in Table 3. The Nawaz, Enscore, Ham (NEH) algorithm is utilized to generate the set of initial solutions [57]. The quality of the solution (objective function) represents the completion time of the last job in the schedule and is calculated using Eq. (5) [57]. Let $\pi(q)$ be the index of the job assigned to qth place in the permutation of jobs (π). $start_{\pi(q),j}$ is the starting time of qth job on machine j and $p_{\pi(q),j}$ the processing time of the qth job on machine j, then:

$$start_{\pi(q),j} = max\ \{start_{\pi(q),j-1}, start_{\pi(q-1),j}\}$$

with $start_{\pi(0),j} = 0$ and $start_{\pi(q),0} = 0$ and its completion time is calculated as (Eq. (5)):

$$C_{\pi(q),j} = start_{\pi(g)} + p_{\pi(q),j} \tag{5}$$

Table 1
The MAX-SAT instances.

#	Instance	Name	Variables	Clauses
1	SAT 1	parity-games/instance-n3-i3-pp	525	2276
2	SAT 2	parity-games/instance-n3-i4-pp-ci-ce	696	3122
3	SAT 3	parity-games/instance-n3-i3-pp-ci-ce	525	2336
4	SAT 4	jarvisalo/eq.atree.braun.8.unsat	684	2300
5	SAT 5	highgirth/3SAT/HG-3SAT-V300-C1200-4	300	1200

Table 2
The one dimensional bin packing instances.

#	Instance	Name	Capacity	No. pieces
1	BP 1	triples2004/instance1	1000	2004
2	BP 2	falkenauer/u1000-01	150	1000
3	BP 3	test/testdual7/binpack0	100	5000
4	BP 4	50–90/instance1	150	2000
5	BP 5	test/testdual10/binpack0	100	5000

Table 3
The permutation flow shop instances.

#	Instance	Name	No. jobs	No. machines
1	FS 1	100 × 20/2	100	20
2	FS 2	500 × 20/2	500	20
3	FS 3	100 × 20/4	100	20
4	FS 4	200 × 20/1	200	20
5	FS 5	500 × 20/3	500	20

Table 4
The personnel scheduling problem instances.

#	Instance	Name	Staff	Shift types	Days
1	PS 1	Ikegami-3Shift-DATA1.2	25	3	30
2	PS 2	MER-A	54	12	42
3	PS 3	ERRVH-B	51	8	42
4	PS 4	BCV-A.12.1	12	5	31
5	PS 5	ORTEC01	16	4	31

Table 5
The traveling salesman problem instances.

#	Instance	Name	No. cities
1	TSP 1	pr299	299
2	TSP 2	usa13509	13,509
3	TSP 3	rat575	575
4	TSP 4	u2152	2152
5	TSP 5	d1291	1291

Table 6
The vehicle routing problem instances.

#	Instance	Name	No. vehicles	Vehicle capacity
1	VRP 1	Homberger/RC/RC2-10-1	250	1000
2	VRP 2	Solomon/RC/RC103	25	200
3	VRP 3	Homberger/C/C1-10-1	250	200
4	VRP 4	Solomon/R/R101	25	1000
5	VRP 5	Homberger/RC/RC1-10-5	250	200

4.1.4. Problem domain IV: Personnel scheduling

In personnel scheduling, given a set of employees of specific categories, a set of pre-defined periods (shifts) on a working day, and a set of working days; schedule each employee to specific planning periods in such a way that the assignment satisfies the operational requirements [19]. The goal is to satisfy a range of preferences as far as possible. Table 4 shows the characteristics of the problem instances. The set of initial solutions are generated using a neighborhood operator based method which gradually adds new shifts to the roster until all employees have been scheduled. The quality of the solutions is assessed based on how many soft constraints (preferences) are satisfied. See [19] for more details.

4.1.5. Problem domain V: Traveling salesman

In the traveling salesman problem, given a set of cities and their positions (pairwise distances), find the shortest path where each city is visited only once and the path ends at the starting city [3]. The aim is to minimize the total traveling distance. Table 5 shows the instances that we use. The set of initial solutions are generated by randomly generating permutation sequences. The quality of the solutions is represented by the total traveling distance of the solution route (Eq. (6)) [3]. Let $x_{ij} = 1$ if the path goes from city i to city j and zero otherwise, c_{ij} is the distance between city i and city j, and n is the number of cities. The total traveling distance (objective function) of a given solution is calculated as follows (Eq. (6)) [3]:

$$f = \sum_{i=1}^{n} \sum_{i \neq j, j=0}^{n} c_{ij} x_{ij} \tag{6}$$

4.1.6. Problem domain VI: The vehicle routing with time windows

We are given a set of customers, each one with a known demand and service time, and a set of vehicles with a known maximum capacity. We have to design a set of routes to serve all customers in such a way that each vehicle starts and ends

at the depot, the total demand of each route does not exceed the vehicle capacity and each customer is visited exactly once by exactly one vehicle during its time window(s) [56]. The aim is to minimize the total traveling distance. The characteristics of the instances are presented in Table 6. The set of initial solutions are generated as follows: first create an empty route, then iterate through all customers and add any one to the current route that does not violate any constraints. If no customer can be added to the current route, create a new route. The process is repeated until all customers have been assigned to a route. The quality of a solution (objective function), representing the number of vehicles needed and the total traveling distance, is calculated using Eq. (7) [56]:

$$f = n * C + \sum_{i=1}^{n} d_i \tag{7}$$

where n is the total number of vehicles, C is a constant fixed to 1000 [7,38] and d represents the distance travelled by ith vehicle. Please note that Eq. (7) balances the two objectives of minimizing the number of vehicles and the total traveled distance. The constant C is used to give more importance to the number of vehicles in a given solution. See [56] for more details.

4.1.7. HyFlex problem specific heuristics: The perturbative low level heuristics

For each of the HyFlex problem domains, a set of different perturbative low level heuristics are provided [7]. The low level heuristics are categorized into four types as follows [7]:

1. Mutational or perturbation heuristics: perturb the current solution to generate a new solution by removing, swapping, adding or deleting one solution component. The mutation intensity is controlled by α, $0 \leqslant \alpha \leqslant 1$, which represents the fraction of the total number of variables that are to be changed by the mutation operator.
2. Destruction–construction heuristics: destroy a part of the current solution and reconstruct it to generate a new solution. The difference between destruction–construction and mutational heuristics is that the destruction–construction can be seen as large neighborhood structures and they use problem specific construction heuristics to recreate the solutions.
3. Hill-climbing or local search heuristics: given an initial solution, iteratively generate a neighborhood solution, only accepting improving solutions, until a stopping condition is satisfied. The depth of the search is controlled by β which takes a value in the range [0,1] and represents the number of improving steps to be completed by the local search heuristics (only first improvement solution is accepted). The hill-climbing heuristic is different from mutational heuristic as it an iterative improvement process and accepts only improving solutions.
4. Crossover heuristics: given two solutions, combine them to generate a new one. Examples of implemented crossover operators are: one point crossover, two point crossover, exon shuffling crossover, partially mapped crossover and order crossover [7].

Table 7 presents the total number of each type of the perturbative low level heuristics for the six HyFlex problem domains [7].

4.2. MCTS-HH parameter settings

The proposed MCTS-HH has six different parameters that need to be set in advance. The nature of the HyFlex problem domains, in terms of the objective function, constraints and the landscape difficulty vary from one problem to another. As a result, parameter settings have a critical impact on MCTS-HH performance. Hence, MCTS-HH parameters need to be calibrated in such a way that the values are not suited to only one problem instance and/or domain. To calibrate the parameters, we randomly selected two instances from each problem domain. The selected instances are: SAT 1 and SAT 4 from the MAX-SAT domain, BP 4 and BP 5 from the bin packing domain, FS 1 and FS 4 from the permutation flow shop domain, PS 2 and PS 5 from the personnel scheduling domain, TSP 4 and TSP 5 from the traveling salesman domain and VRP 3 and VRP 4 from the vehicle routing with time windows domain. Three of MCTS-HH parameters (No. of destruction–construction, depth of search and the mutation intensity) are not calibrated because we used various values for these parameters in order to generate different low level heuristics. We utilize the Relevance Estimation and Value Calibration method (REVAC) [40] to find the generic values that can be used across all problem domains, instead of using different values for each domain/instance.

Table 7
The HyFlex low level heuristic types (M: mutation. R&R: Ruin-recreate. HC: Hill-climbing. Xover: Crossover).

#	Problem domains	M	R&R	HC	Xover	Total
1	Boolean satisfiability	4	1	2	2	9
2	One dimensional bin packing	3	2	2	1	8
3	Permutation flow shop	5	2	4	3	14
4	Personnel scheduling	1	3	4	3	11
5	Traveling salesman	5	1	6	3	15
6	Vehicle routing	4	2	4	2	12

Table 8
MCTS-HH parameter settings.

#	Parameters	Possible range	Suggested value
1	Tree depth, n	1–15	3
2	Tree width, m	1–15	3
3	Memory size	2–10	8
4	The scaling factor c	2–20	12
5	No. of destruction–construction	–	1–20
6	Depth of search	–	[0,1]
7	Mutation intensity	–	[0,1]

REVAC is a tool for parameter optimization that is based on a steady state genetic algorithm and entropy theory. The running time for each instance is fixed to 10 s and the number of iterations performed by REVAC is fixed at 100 iterations (see [40] for more details). Then the average of the average value for each parameter value obtained by REVAC is used for all tested instances by MCTS-HH. Table 8 lists the parameter settings of MCTS-HH that have been used for all problem domains.

5. Computational results and discussion

Adhering to CHeSC rules [7], MCTS-HH is executed for 31 independent runs using different initial solutions and random seeds. For each run, the execution time is used as the stopping condition, which is determined by benchmark software provided by the CHeSC organizers to facilitate a fair comparison between different researchers that use different machines. For each instance, the best and the median results of MCTS-HH over 31 runs are reported. In addition, the percentage deviation from the best known value obtained in the hyper-heuristics literature is also calculated for each instance: $\Delta(\%) = ((best_{MCTS-HH} - best)/best) * 100$, where $best_{MCTS-HH}$ is the best result returned over 31 runs by MCTS-HH and best is the best result produced by other hyper-heuristics. We also compare MCTS-HH against other hyper-heuristic frameworks using a *Formula one* measure, which calculates the overall rank of the MCTS-HH compared to other hyper-heuristics. This is

Table 9
Best results of MCTS-HH compared to the top five hyper-heuristic frameworks. Note: $\Delta(\%)$ represents the percentage deviation from the best result obtained by other hyper-heuristic methods.

Domain	Instances	MCTS-HH			The top five hyper-heuristic framework from CHeSC competition				
		Best	$\Delta(\%)$	Rank	AdapHH	VNS-TW	ML	PHUNTER	EPH
MAX-SAT	SAT 1	**0**	−100	1	1	1	1	1	4
	SAT 2	**1**	0	=	3	**1**	3	5	5
	SAT 3	**0**	−100	1	1	1	1	2	2
	SAT 4	**1**	0	=	**1**	**1**	4	4	5
	SAT 5	**7**	0	=	9	**7**	**7**	**7**	**7**
Bin packing	BP 1	**0.0131**	0	=	**0.0131**	0.0298	0.0323	0.0397	0.043
	BP 2	**0.0028**	0	=	**0.0028**	0.0036	0.0067	0.0034	0.0034
	BP 3	**0.0004**	0	=	**0.0004**	0.0136	0.0124	0.0178	0.008
	BP 4	**0.1083**	0	=	**0.1083**	0.1087	0.1084	0.1088	0.1083
	BP 5	**0.0031**	0	=	**0.0031**	0.0238	0.0178	0.0318	0.0136
Flow shop	FS 1	**6212**	−0.032	1	6214	6230	6226	6221	6232
	FS 2	**26,720**	−0.067	1	26,757	26,765	26,744	26,786	26,738
	FS 3	**6285**	−0.285	1	6303	6303	6304	6303	6309
	FS 4	11,320	0.017	2	**11,318**	11,333	11,338	11,336	11,328
	FS 5	**26,528**	−0.026	1	26,541	26,535	26,559	26,600	26,569
Personnel scheduling	PS 1	**11**	0	1	17	13	**11**	13	16
	PS 2	**9328**	−0.203	1	9435	9347	9436	9624	9747
	PS 3	**3120**	−0.128	1	3142	3124	3138	3142	3142
	PS 4	**1348**	−1.605	1	1448	1370	1384	1350	1469
	PS 5	**280**	−3.448	1	295	290	300	290	310
Traveling salesman	TSP 1	**48194.9**	0	=	**48194.9**	**48194.9**	**48194.9**	**48194.9**	**48194.9**
	TSP 2	**20410692.5**	−1.648	1	20752853.8	2084855.6	20793219.8	20754199.8	20941645.1
	TSP 3	**6790.5**	−0.08	1	6797.5	6796	6805.3	6796	6799.2
	TSP 4	65961.1	0.0037	2	66277.1	66830.2	66428.2	66641.4	**65958.6**
	TSP 5	52056.8	0.0065	2	52383.8	52896.5	52626.7	52,172	**52053.4**
Vehicle routing	VRP 1	**58050.5**	−0.002	1	58052.1	68340.4	67622.1	61139.3	63932.2
	VRP 2	**12260.1**	−0.023	1	13304.9	13298.1	13298.4	12,263	13,284
	VRP 3	**142,461**	−0.039	1	145481.5	144012.6	142,517	143663.9	143510.8
	VRP 4	20651.6	0.0024	3	20652.3	20651.1	20651.1	**20650.8**	**20650.8**
	VRP 5	**144256.47**	−1.178	1	146,154	146513.6	146200.8	146472.9	145976.5

the same measure used by the CHeSC organizers to rank the competitors. The results of MCTS-HH are compared with the top five hyper-heuristics from the CHeSC competition (http://www.asap.cs.nott.ac.uk/external/chesc2011/raw-results.html). These are:

1. AdapHH: An intelligent hyper-heuristic framework for CHeSC 2011 [37].
2. VNS-TW: A variable neighborhood search-based hyperheuristic for cross-domain optimization problems in the CHeSC 2011 competition [26].
3. ML: A hyper-heuristic for the CHeSC 2011 competition [33].
4. Pearl Hunter: A hyper-heuristic that Compiles Iterated Local Search Algorithms [59].
5. EPH: An evolutionary Programming Hyper-heuristic with Co-evolution for the CHeSC 2011 [36].

The results obtained by MCTS-HH are presented in Table 9, including the best, percentage deviation ($\Delta(\%)$) and instance ranking, along with the best results obtained by the top five hyper-heuristic frameworks. Best results are highlighted in a bold font. MCTS-HH exhibits very good performance compared to the top five hyper-heuristic frameworks. Of the 30 tested instances, MCTS-HH produced new best results for 17 instances, matched the best results on 8 instances and was inferior on only 5 instances.

In Table 10, we provide the median, percentage deviation and instance ranking results achieved by MCTS-HH in comparison with the median results obtained by the top five hyper-heuristic frameworks. Best results are highlighted in a bold font. It is clear from Table 10 that MCTS-HH obtains better median results for 1 instance and ties with other hyper-heuristic frameworks on 8 out of 30 instances. For the other 21 instances, MCTS-HH obtained the second best median results for 12 instances, third best median results for 3 instances, fourth best median results for 2 instances, fifth best median results for 3 instances and sixth best median result for 1 instance. Although MCTS-HH did not manage to obtain the best median results for all instances, the percentage deviation of these instances is, however, relatively small.

We now compare the results of the MCTS-HH against the top five hyper-heuristic frameworks from the CHeSC competition using the Formula one ranking system that was used to rank the competing hyper-heuristic frameworks. Table 11 shows the ranking score (the higher, the better) obtained by MCTS-HH and the top five hyper-heuristics for the HyFlex six problem

Table 10
Median results of MCTS-HH compared to the top five hyper-heuristic frameworks. Note: $\Delta(\%)$ represents the percentage deviation from the median result obtained by other hyper-heuristic methods.

Domain	Instances	MCTS-HH			The top five hyper-heuristic framework from CHeSC competition				
		Best	$\Delta(\%)$	Rank	AdapHH	VNS-TW	ML	PHUNTER	EPH
MAX-SAT	SAT 1	4	33.30	2	**3**	**3**	5	5	7
	SAT 2	**3**	0	=	5	**3**	10	11	11
	SAT 3	**2**	0	=	**2**	**2**	3	4	6
	SAT 4	**3**	0	=	**3**	**3**	9	9	15
	SAT 5	**8**	0	=	**8**	10	**8**	**8**	13
Bin packing	BP 1	0.0165	2.48	2	0.0161	**0.037**	0.0421	0.0479	0.0504
	BP 2	**0.0036**	0	=	**0.0036**	0.0072	0.0075	0.0036	0.0036
	BP 3	0.0039	8.33	2	**0.0036**	0.0167	0.0146	0.0201	0.0113
	BP 4	**0.1083**	0	=	**0.1083**	0.1088	0.1085	0.1091	0.1087
	BP 5	0.0048	37.14	2	**0.0035**	0.0278	0.0218	0.0395	0.0224
Flow shop	FS 1	6248	0.12	3	**6240**	6251	6245	6253	6250
	FS 2	26,840	0.14	5	26,814	26,803	**26,800**	26,858	26,816
	FS 3	6334	0.17	4	6326	6328	**6323**	6350	6347
	FS 4	11,362	0.02	2	**11,359**	11,376	11,384	11,388	11,397
	FS 5	26,621	0.07	3	26,643	**26,602**	26,610	26,677	26,640
Personnel scheduling	PS 1	19	5.55	2	24	19	**18**	25	22
	PS 2	9631	0.03	2	9667	**9628**	9812	10,136	10,074
	PS 3	3228	0.15	2	3289	**3223**	3228	3255	3232
	PS 4	**1557**	−2.07	1	1765	1590	1605	1595	1615
	PS 5	**315**	0	=	325	320	**315**	320	345
Traveling salesman	TSP 1	**48194.9**	0	=	**48194.9**	**48194.9**	**48194.9**	**48194.9**	**48194.9**
	TSP 2	20826402.8	0.02	2	**20822145.7**	21042675.8	21093828.3	21246427.7	21064606.3
	TSP 3	6819.6	0.13	5	**6810.5**	6819.1	6820.6	6813.6	6811.9
	TSP 4	66983.6	0.34	4	66879.8	67,378	66,894	67136.8	**66756.2**
	TSP 5	52989.7	0.12	3	53099.8	54028.6	54368.4	52934.4	**52925.3**
Vehicle routing	VRP 1	60928.4	0.04	2	**60900.6**	76147.1	80671.3	64717.8	74715.8
	VRP 2	12314.9	0.2	2	13347.6	13367.9	13329.8	**12,290**	13335.6
	VRP 3	145390.3	0.03	2	148516.8	148206.2	**145333.5**	146944.4	162188.5
	VRP 4	20,684	0.16	6	20656.6	21642.9	20654.1	**20650.8**	**20650.8**
	VRP 5	148988.9	0.22	5	148689.2	149132.4	148975.1	**148,659**	155224.7

Table 11
The ranking of MCTS-HH and the top five hyper-heuristics.

#	Algorithms	SAT	BP	FS	PS	TSP	VRP	Overall score
1	MCTS-HH	***32.10***	34.00	25.00	***37.33***	33.00	23.00	***184.43***
2	AdapHH	31.60	***44.00***	36.00	7.00	***37.75***	14.00	170.35
3	VNS-TW	31.60	2.00	31.00	35.50	13.75	4.00	117.85
4	ML	10.50	8.00	***38.00***	28.33	11.00	21.00	116.83
5	PHUNTER	7.50	2.00	6.00	9.50	23.75	***30.00***	78.75
6	EPH	0.00	6.00	17.00	7.50	33.75	11.00	75.25

domains (MAX-SAT, BP, FS, PS, TSP and VRP). Last column (Table 11) indicates the overall rank. Best results are highlighted in a bold font.

Considering the results in Table 11, we can see that, MCTS-HH obtained the first, second, third, first, third and second rank for the SAT, BP, FS, PS, TSP and VRP domains, respectively. Considering the overall rank (last column in Table 11), it is interesting to note that MCTS-HH obtained the first rank compared to the top five hyper-heuristic frameworks. The results in Table 11 also demonstrate that, for all the problem domains, MCTS-HH's score is higher than 20, whilst the scores of the top five hyper-heuristic frameworks vary considerably from one domain to another. These results are evidence that MCTS-HH obtained good results and generalized well over all the HyFlex problem domains, compared to the top five hyper-heuristic methods. The computational results indicate that MCTS-HH produces competitive results, if not better, than other hyper-heuristics on some instances. MCTS-HH also generalizes well over all the HyFlex problem domains, when compared to the top five hyper-heuristic frameworks in the current scientific literature.

6. Discussion

The numerical results presented throughout this work demonstrate that, across six different combinatorial optimization problems, MCTS-HH achieved favorable results compared to the top five hyper-heuristic frameworks from the CHeSC competition. More importantly, MCTS-HH obtained new best results for 17 out of 30 instances. In all domains, the percentage deviation of MCTS-HH reveals that MCTS-HH results are stable and very close to the best results obtained by other hyper-heuristic frameworks. These results are also supported by the ranking system. We believe that we have demonstrated that MCTS-HH works well across six different combinatorial optimization problems. The good results, we believe, are due to the following two factors:

1. The ability of the proposed MCTS-HH in handling the search space of the low level heuristics and identifying the best sequence of heuristics to be applied at the current decision point. By focusing on a specific area of the low level heuristic search space, MCTS-HH can exclude the poor performing low level heuristics and thus only good low level heuristics are applied.
2. The ability of the proposed MCTS-HH in generating, for each instance, different sequences of low level heuristics during the search. By generating, for each instance, a different sequence of heuristics, the hyper-heuristic can deal with changes that might occur during the search, escaping from the local optima and exploring different areas in the problem solution search space.

7. Conclusions

In this work, we have proposed a Monte Carlo tree search hyper-heuristic framework for optimization problems. To deal with various instance characteristics, the proposed framework utilizes a large set of heuristics and models the heuristic search space as a tree. The proposed framework uses the Monte Carlo tree search algorithm to search the tree, evolving a sequence of low level heuristics to be applied to the given instance solution. To diversify the search process, we have embedded the proposed hyper-heuristic with a memory mechanism, which contains a population of solutions, using different population updating rules. The generality of the proposed framework is demonstrated using the six challenging problems from the hyper-heuristic competition (CHeSC) test suite. The experimental results demonstrate that the proposed framework produces highly competitive results, if not superior to the current state of the art. It also generalizes well over all six problem domains when compared to the top five hyper-heuristic frameworks from the CHeSC competition.

References

[1] M. Ayob, G. Kendall, A monte carlo hyper-heuristic to optimize component placement sequencing for multi head placement machine, in: Proceedings of the International Conference on Intelligent Technologies (InTech'03), 2003, pp. 132–141.
[2] R. Bai, J. Blazewicz, E.K. Burke, G. Kendall, B. McCollum, A simulated annealing hyper-heuristic methodology for flexible decision support, 4OR – Quart. J. Oper. Res. 10 (1) (2012) 43–66.
[3] M. Bellmore, G.L. Nemhauser, The traveling salesman problem: a survey, Oper. Res. 16 (3) (1968) 538–558.

[4] B. Bilgin, P. Demeester, M. Misir, W. Vancroonenburg, G. Vanden Berghe, One hyper-heuristic approach to two timetabling problems in health care, J. Heuristics 18 (3) (2012) 401–434.
[5] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A survey of monte carlo tree search methods, IEEE Trans. Comput. Intell. AI Games 4 (1) (2012) 1–43.
[6] E. Burke, M. Dror, S. Petrovic, R. Qu, Hybrid graph heuristics within a hyper-heuristic approach to exam timetabling problems, in: Next Wave in Computing, Optimization, and Decision Technologies: 9th INFORMS-Computing-Society Conference, Operations Research/Computer Science Interfaces Series, 2005, pp. 79–91.
[7] E. Burke, M. Gendreau, M. Hyde, G. Kendall, B. McCollum, G. Ochoa, A. Parkes, S. Petrovic, The cross-domain heuristic search challenge – an international research competition, in: X. Yao, C.A.C. Coello (Eds.), Proceedings of Learning and Intelligent Optimization (LION5), Lecture Notes in Computer Science, vol. 6683, 2011, pp. 631–643.
[8] E. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, J. Woodward, A classification of hyper-heuristic approaches, in: Chapter Handbook of Meta-heuristics, International Series in Operations Research & Management Science, 2010, pp. 449–468.
[9] E. Burke, M. Hyde, G. Kendall, J. Woodward, Automating the packing heuristic design process with genetic programming, Evolut. Comput. 20 (1) (2012) 63–89.
[10] E. Burke, G. Kendall, E. Soubeiga, A tabu-search hyperheuristic for timetabling and rostering, J. Heuristics 9 (6) (2003) 451–470.
[11] E.K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: a survey of the state of the art, J. Oper. Res. Soc. 64 (12) (2013) 1695–1724.
[12] S. Cai, K. Su, Local search for boolean satisfiability with configuration checking and subscore, Artif. Intell. 204 (2013) 75–98.
[13] F. Caraffini, F. Neri, G. Iacca, A. Mol, Parallel memetic structures, Inf. Sci. 227 (2013) 60–82.
[14] F. Caraffini, F. Neri, L. Picinali, An analysis on separability for memetic computing automatic design, Inf. Sci. 265 (2014) 1–22.
[15] P. Cowling, G. Kendall, L. Han, An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem, in: Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002), Hilton Hawaiian Village Hotel, Honolulu, Hawaii, 12–17 May, 2002a, pp. 1185–1190.
[16] P. Cowling, G. Kendall, E. Soubeiga, A hyperheuristic approach to scheduling a sales summit, in: E. Burke, W. Erben (Eds.), Practice and Theory of Automated Timetabling III, Lecture Notes in Computer Science, vol. 2079, 2000, pp. 176–190.
[17] P. Cowling, G. Kendall, E. Soubeiga, Hyperheuristics: a tool for rapid prototyping in scheduling and optimisation, in: S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, G. Raidl (Eds.), Applications of Evolutionary Computing: Proceedings of Evo Workshops 2002, 3–4 April, Lecture Notes in Computer Science, vol. 2279, Springer-Verlag, Kinsale, Ireland, 2002, pp. 269–287.
[18] W. Crowston, F. Glover, G.T.T. Trawick, Probabilistic and parametric learning combinations of local job shop scheduling rules, in: ONR Research memorandum, GSIA, vol. 1, no. 117, Carnegie Mellon University, Pittsburgh, 1963.
[19] T. Curtois, G. Ochoa, M. Hyde, J.A. Vazquez-Rodriguez, A HyFlex Module for the Personnel Scheduling Problem, Technical Report, School of Computer Science, University of Nottingham, 2010.
[20] J. Denzinger, M. Fuchs, Goal oriented equational theorem proving using team work, in: Proceedings of KI-94: Advances in Artificial Intelligence: 18th German Annual Conference on Artificial Intelligence Saarbrcken, Germany, September 18–23, Lecture Notes in Artificial Intelligence, vol. 861, 1994, pp. 343–354.
[21] J. Denzinger, M. Fuchs, High Performance ATP Systems by Combining Several AI Methods, Technical Report, SEKI-Report SR-96-09, University of Kaiserslautern, 1996.
[22] J. Denzinger, M. Fuchs, M. Fuchs, High performance ATP systems by combining several AI methods, in: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97), 1997, pp. 102–107.
[23] H. Dixon, M. Ginsberg, A. Parkes, Generalizing boolean satisfiability i: Background and survey of existing work, J. Artif. Intell. Res. 21 (2004) 193–243.
[24] H.L. Fang, P. Ross, D. Corne, A promising hybrid ga/heuristic approach for open-shop scheduling problems, in: Proc. 11 th European Conference on Artificial Intelligence, 1994, pp. 590–594.
[25] H. Fisher, G. Thompson, Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules, Chapter Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules, Industrial Scheduling, Prentice-Hall, Englewood Cliffs, NJ, 1963. pp. 225–251.
[26] P.-C. Hsiao, T.-C. Chiang, L.-C. Fu, A Variable Neighborhood Search-Based Hyperheuristic for Cross-Domain Optimization Problems in CHeSC 2011 Competition, Technical Report, School of Computer Science, University of Nottingham, 2011.
[27] M. Hyde, G. Ochoa, T. Curtois, J.A. Vazquez-Rodriguez, A HyFlex Module for the One Dimensional Bin Packing Problem, Technical Report, School of Computer Science, University of Nottingham, 2010.
[28] M.H. Hyde, E.K. Burke, G. Kendall, Automated code generation by local search, J. Oper. Res. Soc. 64 (12) (2013) 1725–1741.
[29] G. Iacca, F. Neri, E. Mininno, Y.-S. Ong, M.-H. Lim, Ockham's razor in memetic computing: three stage optimal memetic exploration, Inf. Sci. 188 (2012) 17–43.
[30] G. Kendall, E. Burke, Hyper-heuristics, Chapter Wiley Encyclopedia of Operations Research and Management Science, Wiley, 2011.
[31] G. Kendall, J. Li, Competitive travelling salesmen problem: a hyper-heuristic approach, J. Oper. Res. Soc. 64 (2) (2013) 208–216.
[32] G. Kendall, N. Mohd Hussin, A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology, in: E. Burke, M. Trick (Eds.), Lecture Notes in Computer Science: Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, vol. 3616, 2005, pages 270–293.
[33] M. Larose, A Hyper-Heuristic for the CHeSC 2011, Technical Report, School of Computer Science, University of Nottingham, 2011.
[34] S. Martello, P. Toth, Knapsack Problems, John Wiley and Sons, Chichester, UK, 1990.
[35] M. Mashael, E. Özcan, G. Kendall, A multi-objective hyper-heuristic based on choice function, Expert Syst. Appl. 41 (9) (2014) 4475–4493.
[36] D. Meignan, An Evolutionary Programming Hyper-Heuristic with Co-evolution for CHeSC'11, Technical Report, School of Computer Science, University of Nottingham, 2011.
[37] M. Misir, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, An intelligent hyper-heuristic framework for CHeSC 2011, in: Y. Hamadi, M. Schoenauer (Eds.), Learning and Intelligent Optimization: 6th International Conference, LION 6, Paris, France, January 16–20, 2012, Revised Selected Papers, 2012, pp. 461–466.
[38] M. Misir, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, A new hyper-heuristic as a general problem solver: an implementation in HyFlex, J. Sched. 16 (3) (2013) 291–311.
[39] M. Misr, Verbeeck, P. De Causmaecker, G. Vanden Berghe, An investigation on the generality level of selection hyper-heuristics under different empirical conditions, Appl. Soft Comput. 13 (7) (2013) 3335–3353.
[40] V. Nannen, A. Eiben, Efficient relevance estimation and value calibration of evolutionary algorithm parameters, in: Proceedings of the IEEE Congress of Evolutionary Computation (CEC 2007), 2007, pp. 103–110.
[41] F. Neri, C. Cotta, Memetic algorithms and memetic computing optimization: a literature review, Swarm Evolut. Comput. 2 (2012) 1–14.
[42] F. Neri, C. Cotta, P. Moscato, Handbook of Memetic Algorithms, vol. 379, Springer, 2011.
[43] F. Neri, E. Mininno, G. Iacca, Compact particle swarm optimization, Inf. Sci. 239 (2013) 96–121.
[44] Y.S. Ong, A.J. Keane, Meta-lamarckian learning in memetic algorithms, IEEE Trans. Evol. Comput. 8 (2) (2004) 99–110.
[45] Y.-S. Ong, M.-H. Lim, N. Zhu, K.-W. Wong, Classification of adaptive memetic algorithms: a comparative study, IEEE Trans. Syst. Man Cybern. Part B: Cybern. 36 (1) (2006) 141–152.
[46] E. Özcan, B. Bilgin, E. Korkmaz, Hill climbers and mutational heuristics in hyperheuristics, in: Parallel Problem Solving from Nature – PPSN IX, Lecture Notes in Computer Science, vol. 4193, 2006, pp. 202–211.

[47] P. Rattadilok, A. Gaw, R. Kwan, Distributed choice function hyper-heuristics for timetabling and scheduling, in: E. Burke, M. Trick (Eds.), Lecture Notes in Computer Science: Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, vol. 3616, 2005, pp. 51–67.
[48] A. Rimmel, F. Teytaud, C.-S. Lee, S.-J. Yen, M.-H. Wang, S.-R. Tsai, Current frontiers in computer go, IEEE Trans. Comput. Intell. AI Games 2 (4) (2010) 229–238.
[49] P. Ross, Hyper-Heuristics, Chapter Introductory Tutorials in Optimization and Decision Support Techniques, Springer, 2014. pp. 611–638.
[50] N. Sabar, M. Ayob, Examination timetabling using scatter search hyper-heuristic, in: 2nd Conference on Data Mining and Optimization, 2009, pp. 127–131.
[51] N. Sabar, M. Ayob, Kendall, R. Qu, Grammatical evolution hyper-heuristic for combinatorial optimization problems, IEEE Trans. Evol. Comput. 16 (6) (2013) 840–861.
[52] N. Sabar, M. Ayob, R. Qu, G. Kendall, A graph coloring constructive hyper-heuristic for examination timetabling problems, Appl. Intell. 37 (1) (2012) 1–11.
[53] A. Soghier, R. Qu, Adaptive selection of heuristics for assigning time slots and rooms in exam timetables, Appl. Intell. 39 (2) (2013) 438–450.
[54] K. Sörensen, Metaheuristics – the metaphor exposed. Int. Trans. Oper. Res. (2013), http://dx.doi.org/10.1111/itor.12001.
[55] E. Taillard, Benchmarks for basic scheduling problems, J. Oper. Res. Soc. 64 (2) (1993) 278–285.
[56] P. Toth, D. Vigo (Eds.), The Vehicle Routing Problem, Discrete Mathematics and Applications, SIAM, 2002.
[57] J.A. Vazquez-Rodriguez, G. Ochoa, T. Curtois, M. Hyde, A HyFlex Module for the Permutation Flow Shop Problem, Technical Report, School of Computer Science, University of Nottingham, 2010.
[58] D. Wolpert, W. Macready, No free lunch theorems for optimization, IEEE Trans. Evolut. Comput. 1 (1) (1997) 67–82.
[59] F. Xue, C. Chan, W. Ip, C. Cheung, Pearl Hunter: A Hyper-Heuristic that Compiles Iterated Local Search Algorithms, Technical Report, School of Computer Science, University of Nottingham, 2011.