Information Sciences 436–437 (2018) 89–107

Contents lists available at ScienceDirect

Information Sciences

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

Automatic design of hyper-heuristic based on reinforcement learning

Shin Siang Choong[a], Li-Pei Wong[a,*], Chee Peng Lim[b]

[a] School of Computer Sciences, Universiti Sains Malaysia, Malaysia
[b] Institute for Intelligent Systems Research and Innovation, Deakin University, Australia

ARTICLE INFO

Article history:
Received 10 November 2016
Revised 26 July 2017
Accepted 4 January 2018
Available online 9 January 2018

Keywords:
Hyper-heuristic
Q-learning
Automatic design
Cross-domain heuristic search

ABSTRACT

Hyper-heuristic is a class of methodologies which automates the process of selecting or generating a set of heuristics to solve various optimization problems. A traditional hyper-heuristic model achieves this through a high-level heuristic that consists of two key components, namely a heuristic selection method and a move acceptance method. The effectiveness of the high-level heuristic is highly problem dependent due to the landscape properties of different problems. Most of the current hyper-heuristic models formulate a high-level heuristic by matching different combinations of components manually. This article proposes a method to automatically design the high-level heuristic of a hyper-heuristic model by utilizing a reinforcement learning technique. More specifically, Q-learning is applied to guide the hyper-heuristic model in selecting the proper components during different stages of the optimization process. The proposed method is evaluated comprehensively using benchmark instances from six problem domains in the Hyper-heuristic Flexible Framework. The experimental results show that the proposed method is comparable with most of the top-performing hyper-heuristic models in the current literature.

1. Introduction

An optimization model involves finding the feasible solutions from a finite set of solutions that exist in the search space, and then identifying only the optimal solution. There are some exact optimization models that guarantee global optimality. However, this guarantee is limited to small problems, and for complex problems, the exact optimization model may take a long time to reach optimality [64]. In this case, the use of heuristics to produce a solution that is good enough for solving the problem in a reasonable timeframe is a viable option. A heuristic involves applying a practical methodology that is not guaranteed to converge to a global optimum, but a sufficiently good solution. In view of the availability of a large variety of problem-specific heuristics, a key question concerning the selection of a particular heuristic has been posed in the literature in recent years. This leads to the studies on the hyper-heuristic methodology.

A hyper-heuristic is a high-level automated methodology for selecting or generating a set of heuristics [7]. The term "hyper-heuristic" was coined by Denzinger et al. [16]. Fig. 1 shows a classification of hyper-heuristic approaches. There are two main hyper-heuristic categories, i.e. selection hyper-heuristic and generation hyper-heuristic [8]. These two categories can be defined as 'heuristics to select heuristics' and 'heuristics to generate heuristics' respectively [7]. The heuristics to be selected or generated in a hyper-heuristic model are known as the low-level heuristics (LLHs). Both selection and generation

* Corresponding author.
E-mail address: lpwong@usm.my (L.-P. Wong).

https://doi.org/10.1016/j.ins.2018.01.005
0020-0255/

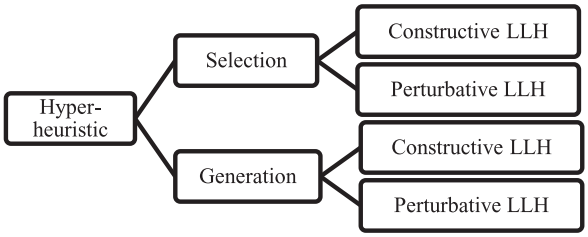

Fig. 1. A classification of hyper-heuristic models.

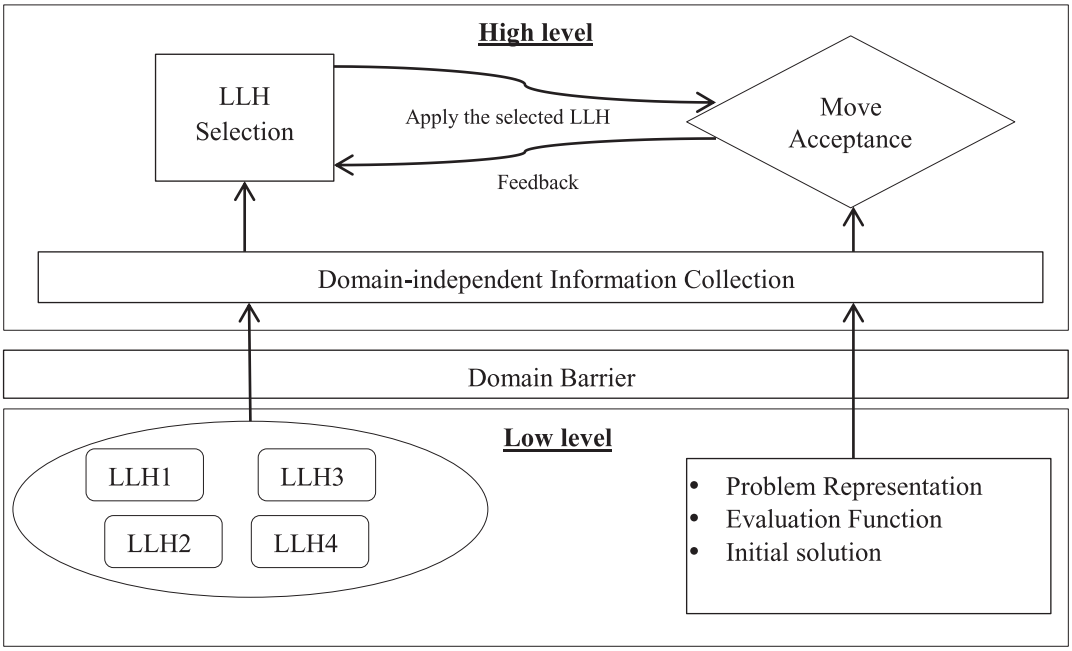

Fig. 2. A generic structure of a traditional hyper-heuristic model.

hyper-heuristics can be further divided into two categories based on the nature of the LLHs [8], namely either constructive or perturbative hyper-heuristics. A constructive hyper-heuristic incrementally builds a complete solution from scratch. On the other hand, a perturbative hyper-heuristic iteratively improves an existing solution by performing its perturbative mechanisms. According to Ochoa et al. [50], perturbative LLHs can be further categorised into ruin-recreate heuristics, mutation heuristics, crossover heuristics, and hill-climbing heuristics. All these LLHs are problem specific, e.g. for the Traveling Salesman Problem, the perturbative LLHs include swap mutation, insert mutation, order crossover, partially mapped crossover, 2-opt local search, 3-opt local search.

A traditional selection hyper-heuristic model consists of two levels, as shown in Fig. 2 [7]. The low level contains a representation of the problem, evaluation function(s), and a set of problem specific LLHs. The high level manages which LLH to use for producing a new solution(s), and then decides whether to accept the solution(s). Therefore, the high-level heuristic performs two separate tasks i.e. (i) LLH selection and (ii) move acceptance [51]. The LLH selection method is a strategy to select an appropriate perturbative LLH to modify the current solution and the move acceptance method decides whether to accept the newly generated solution. Both LLH selection and move acceptance methods have a dramatic impact on the performance of the hyper-heuristic model. Their effects are problem-dependent as different problem domains, or even different instances in a single domain, have different fitness landscape properties [56]. Choosing suitable LLH selection and move acceptance methods for a particular problem is a non-trivial task during the process of designing a robust hyper-heuristic model.

There are many different pairwise combinations of LLH selection and move acceptance methods in the literature [7]. However, in most of the existing hyper-heuristic models, the high-level heuristic is designed manually [56]. In such manual design, the concern is to determine which combination of the LLH selection and move acceptance methods is the best for a particular problem. One of the most straightforward methods is trial-and-error to find the best combination for each problem. However, this is extremely time consuming as the possible number of combinations of these methods is large. In

addition, the optimal configuration of these methods varies during different stages of the optimization process. As a result, an effective way that can automatically design a hyper-heuristic model is necessary.

This research focuses on automating the design process of hyper-heuristic models. Specifically, Reinforcement Learning (RL) [62] is investigated to intelligently select the appropriate LLH selection and move acceptance methods for different stages of the optimization process. RL is a learning algorithm that selects a suitable action based on experience by interacting with the environment. The learner receives a reward or penalty after performing each selected action. As such, it learns which action to perform by evaluating the action choices through cumulative rewards. In the context of automatically designing a hyper-heuristic model, the combinations of the high-level heuristic components (i.e. LLH selection and move acceptance methods) denote the action choices. The RL algorithm rewards or penalizes each combination of the components based on its performance during the optimization process. There are a few RL algorithms that have been extensively applied as feedback mechanisms to tackle decision making problems e.g. Dynamic Programming (DP) [15], SARSA [63], and Q-learning [67].

Q-learning is proposed to automatically design the hyper-heuristic model during different stages of the optimization process in this article. The resulting Q-learning based hyper-heuristic model is denoted as QHH. In QHH, different criteria in selecting the LLH and accepting a proposed solution are used at different intervals. As an example, a criterion at time t may lead to selecting an LLH that has a fast speed in creating a new solution, while at time t', a criterion may lead to choosing an LLH that results in a better performance. To demonstrate the effectiveness of QHH, benchmark instances from six problem domains in the Hyper-heuristic Flexible (HyFlex) framework [50] is used, and the results indicate the competitiveness of QHH.

The remaining of this article contains a description of related work in Section 2, which covers a review on hyper-heuristics, reinforcement learning for hyper-heuristics, and automatic design of hyper-heuristics. Q-learning and its applications are discussed in Section 3. Section 4 presents the proposed QHH model. The results and findings including performance comparison are described in Section 5. Finally, concluding remarks are presented in Section 6.

2. Related work

In this section, a review on hyper-heuristics is presented in Section 2.1. Section 2.2 highlights the applications of RL to hyper-heuristics, while Section 2.3 describes a number of methods to automatically design hyper-heuristics.

2.1. Hyper-heuristics

In general, hyper-heuristics are classified into two categories based on the nature of the heuristic search space, namely selection hyper-heuristics and generation hyper-heuristics [8]. A generation hyper-heuristic aims to generate new LLHs by combining the existing heuristic components. On the other hand, a selection hyper-heuristic intelligently selects which LLH to be applied at a given time from a set of human-designed heuristics. This section focuses on the selection hyper-heuristics, which is the proposed approach in this article.

Most of the existing hyper-heuristic literature belongs to the class of selection hyper-heuristics [56]. The motivation of this class of hyper-heuristics is to combine the advantages of multiple LLHs by applying each of them in a relatively suitable situation. A traditional selection hyper-heuristic model consists of two components: the LLH selection method and the move acceptance method [51]. Many combinations of heuristic selection and move acceptance methods have been reported in the literature. The available LLH selection methods include Simple Random, Choice Function, Tabu Search, Harmony Search, and Reinforcement Learning, while the move acceptance methods include Only Improvement, All Moves, Simulated Annealing, Late Acceptance. Tables 1 and 2 summarise these existing LLH selection and move acceptance methods, respectively.

There is a software framework for the development of selection hyper-heuristics, namely HyFlex [50]. It provides domain-specific low-level components such as problem representations, evaluation functions, sets of problem specific LLHs for six different combinatorial optimization problems, i.e. Boolean Satisfiability (SAT), One-dimensional Bin-Packing (BP), Permutation Flow Shop (FS), Personnel Scheduling (PS), Traveling Salesman Problem (TSP) and Vehicle Routing Problem (VRP). HyFlex does not provide access to the domain-specific components. It features a common software interface which permits a fair performance comparison for multiple hyper-heuristic models in six problem domains. The LLHs in HyFlex can be divided into four categories: ruin-recreate heuristics, mutation heuristics, crossover heuristics, and hill-climbing heuristics. HyFlex has been applied to support the cross-domain heuristic search challenge (CHeSC) [29]. After the competition, a growing number of studies that use the 20 CHeSC contestants as a benchmark to evaluate the performance of new selection hyper-heuristics emerge [2,21,58].

Many of the top-performing hyper-heuristic solutions in CHeSC, such as VNS-TW [26], ML [42] and PHUNTER [70], follow an Iterated Local Search (ILS) [43] procedure. In ILS, the incumbent solution iteratively goes through a diversification phase and an intensification phase before an acceptance decision is made. During the diversification phase, a new solution is proposed by performing a perturbation to the incumbent solution. In this phase, only non-greedy LLHs that make changes to the solution with no guarantee of improvement can be applied. In HyFlex, this corresponds to the ruin-recreate heuristics, mutation heuristics, and crossover heuristics categories. After the diversification phase, the intensification phase is initiated to perform local search based on the newly proposed solution. In this phase, greedy LLHs from the hill-climbing heuristics category are considered. Only the solutions that have equal or better fitness than the input solution are returned.

Table 1
Description of existing LLH selection methods.

LLH selection methods	Related works	Description
Simple random	[13,31]	It uniformly selects at random one LLH from a set of available LLH alternatives during each point of the search process.
Choice function	[12,13,19,18]	The performance score of each LLH is calculated by three different measurements, i.e. f_1, f_2, and f_3. Note that f_1 represents the recent performance of every single LLH; f_2 reflects the dependencies between consecutive pairs of LLHs; f_3 records the elapsed time (in seconds) since the last execution of a particular LLH. Some variants in terms of the formulae used appear in different studies.
Tabu search	[40,71]	It maintains a tabu list of LLHs to be excluded for selection during certain number of iterations. Many different conditions to add an LLH into the tabu list have been proposed, e.g. when a LLH results in a worse solution, when a local search heuristic produces an identical solution.
Harmony search	[14]	This is a meta-heuristic method motivated by the music improvisation process. A number of vectors that reflect a sequence of LLHs and their respective parameters are maintained in the Harmony Memory (HM). During each iteration, a new vector is produced either using the existing information in the HM, or randomly. The quality of the newly produced vector is measured by executing it on a solution. If a better performance is obtained, the worst vector in the HM is replaced.
Ant-inspired algorithms	[11,34,36]	The collective behaviours of ants can be computationally realized as algorithmic procedures to solve various optimization problems. This class of algorithms is usually equipped with state transition rules and pheromone update strategies. Some ant-inspired algorithms such as Ant Colony Optimization (ACO), and Ant System with Q-learning (Ant-Q) have been extended as hyper-heuristic such that they can be used to operate on a heuristic search space.
Reinforcement learning	[60,52,17]	This method learns from experience. A score is assigned to each LLH based on its previous performance. If an LLH yields a better move (with respect to some criteria), its score is rewarded. A deteriorated move results in a penalty of its score. The method selects an LLH based on its score.
Multi-Armed Bandit (MAB) based selection	[57,22,21]	LLH selection is modelled as an MAB problem. An MAB problem is a simplified variant of RL, which only involves a single state [62]. MAB involves multiple arms with the reward probability of each arm $\in$ [0, 1]. A gambler selects one arm during each step. Upon each step, the player receives a reward with the reward probability of the selected arm; otherwise, no reward is given for that particular step. In the hyper-heuristic context, the LLHs represent the arms and the bandit algorithm specifies an LLH selection method. A number of variants are presented in Ferreira [21].
Monte Carlo Tree Search (MCTS)	[58]	This method is applied to explore the LLH search space by analysing the most promising moves. The search contains four stages, i.e. selection, expansion, simulation and back-propagation. By selection and expansion, a tree that represents a sequence of LLHs to be applied is generated. In the simulation stage, the LLHs in the generated tree are applied in sequence. Finally, the heuristic value of each node (LLH) in the tree is updated during the back-propagation stage.
Sequence-based selection by a Hidden Markov Model (HMM)	[37]	This is a statistical technique to produce observable sequences. Each LLH represents an action, and the previous applied LLH represents the current state. The current LLH is selected based on a transition probability matrix. The matrix is updated when an improvement pertaining to the global best solution is obtained.

Although the winning solution of CHeSC, namely AdapHH [46], does not strictly follow the ILS model, it uses a number of mechanisms to mimic the properties of an ILS-based hyper-heuristic [18]. The ILS-based hyper-heuristic outperforms the traditional hyper-heuristic that selects and applies an LLH without considering its category, and subsequently makes an acceptance decision [1,51]. Therefore, we use an ILS model in our proposed QHH model.

2.2. Reinforcement learning in hyper-heuristic

A hyper-heuristic can exploit RL when it has a reward/penalty scheme to measure the performance of each LLH [3]. One of the CHeSC contestants, namely KSATS-HH [60], deploys an LLH selection scheme based on RL and Tabu Search. KSATS-HH rewards the LLH that yields an improved move by increasing its score. If an LLH yields a deteriorated move, its score is reset to the minimum, and is eliminated from the selectable LLH pool (i.e. tabu) for the next seven iterations. In Özcan et al. [52], a hyper-heuristic model that applies RL as the LLH selection method and Great Deluge as the move acceptance method was proposed. In this approach, the LLH that yields an improved move is rewarded by an additive method (i.e. increase its score). On the other hand, it has three penalty schemes for an LLH that yields a deteriorated move, namely subtractive, divisional, and root. More examples of RL-based hyper-heuristics can be found in the survey paper by Burke et al. [7].

However, in a strict manner, especially in machine learning research, a method having only a reward/penalty scheme is not sufficient to be categorized as RL [3]. Specifically, RL requires two important features: trial-and-error and delayed reward [62]. The former indicates that a learner is not instructed to perform an action, but discovers which action results in a better reward by trial-and-error. The latter points out that there may be a delay before the effects of the actions appear. Therefore, rewards and penalties may not be issued immediately. Whenever a learner interacts with the environment, it detects a state of the environment. Each state has a set of selectable actions. After selecting and executing an action, a reinforcement signal (i.e. reward/penalty) is generated as a result of the action. There are a number of RL algorithms e.g. DP [15], SARSA [63],

Table 2
Description of existing move acceptance methods.

Move acceptance methods	Related works	Description
Only Improvement (OI)	[10]	It accepts the newly produced solution if it has a better quality than that of the current solution.
All Moves (AM)	[19,18]	It always accepts the newly produced solution.
Metropolis acceptance	[2]	The newly produced solution is accepted with a probability $e^{\Delta f/T}$, where Δf denotes the fitness difference between the newly produced solution and the current solution, and T denotes the temperature. When T is high/low, the chance of accepting a deteriorated move is high/low.
Simulated Annealing (SA)	[60,35]	This is a variant of Metropolis Acceptance. The temperature decreases over time: $e^{\frac{\Delta f}{T} * \frac{t_{max}}{t_{max} - t_{current}}}$, where Δf denotes the fitness difference between the newly produced solution and the current solution, T denotes the initial temperature, t_{max} denotes the maximum allowed time for the algorithm execution, and $t_{current}$ denotes the current elapsed time.
Late Acceptance (LA)	[31,32]	It accepts the newly produced solutions that have better or equal quality as compared with that from the previous n iterations, where n is a positive parameter. During the initial n iterations, every solution that has a better quality than the initial solution is accepted.
Naive Acceptance (NA)	[6]	The improved moves are always accepted, and the deteriorated moves are accepted with 50% probability.

and Q-learning [67]. Although many hyper-heuristics utilizing RL only provide a reinforcement scheme without considering other specialties of RL, there are some "real RL-based hyper-heuristics" that strictly follow the criteria of RL. In Ferreira [21], LLH selection is modelled as an MAB problem, which is a simplified RL scenario with only one single state. A few variants of MAB-based LLH selection methods are presented. Di Gaspero and Urli [17] also proposed a hyper-heuristic model that follows the RL criteria. The effects of a few variants of the immediate reinforcement schemes as well as selection policy and learning functions are investigated in that study. Besides that, it also provides a unique state and action representation.

2.3. Automatic design of hyper-heuristics

The methods to automatically design hyper-heuristics can be categorized in a similar way as those of hyper-heuristics, i.e., automatic selection and automatic generation of high-level heuristic components. Both categories can be further divided to differentiate between online-design and offline-design methods. In the case of online-design, the design process takes places during the real optimization process, i.e. when a problem instance is being solved. If the design process is accomplished before the real optimization process (i.e. by solving a set of training instances), it is considered as an offline-design method.

Adriaensen et al. [1] proposed a semi-automated approach to design a hyper-heuristic model. This approach utilizes an automatic selection methodology. The design of an optimization algorithm is modelled as a meta-optimization problem. Researchers need to make various design decisions when designing the optimization algorithm. Every decision has a set of alternatives. The search space of a meta-optimization problem includes various combinations of the design choices to solve the optimization problem(s), and the objective is to maximize some performance measures. The design decisions included in this approach are LLH selection and move acceptance methods, local search options, and restart conditions. Adriaensen et al. [1] applied a meta-heuristic named FocusedILS which was introduced by Hutter et al. [28] to solve meta-optimization problems. FocusedILS follows an ILS procedure (as described in Section 2.1). During each iteration of FocusedILS, a new solution is created by performing a perturbation and local search procedure to the current solution. If the new solution is fitter than the current one, it replaces the current solution. A candidate solution for the meta-optimization problem represents an optimization algorithm that utilizes one of the combinations of the design choices. The fitness of a candidate solution is evaluated based on the performance pertaining to a set of benchmark problems. For every single fitness evaluation, a full run of an optimization algorithm on a set of training instances is required. Therefore, the meta-optimization process is very time consuming. In this approach, the best method obtained from the meta-optimization process is the Fair Share Iterated Local Search (FSILS) [2]. To show the effectiveness of the automatically-designed algorithm (i.e. FSILS), another set of experiments was conducted using FSILS, and promising results were produced. This approach belongs to the offline-design category because the design (i.e. meta-optimization) is accomplished before the real optimization process. One of the drawbacks of the offline-design methods is they are highly dependent on the generality of the training instances. To generalize the performance of the designed hyper-heuristics to other problems, the training instances need to be sufficiently large. However, a large set of training instances highly increases the computational cost of the design process.

Sabar et al. [56] proposed an approach named GEP-HH for automatically designing hyper-heuristics by using Gene Expression Programming (GEP) [23], which is a variant of Genetic Programming (GP). It is an online rule generation strategy. GEP-HH maintains a population of individuals. Each individual represents a high-level heuristic, i.e. a combination of an LLH selection method and a move acceptance method. The fitness of an individual is evaluated by applying it to a problem for a predetermined number of iterations. The GEP-HH algorithm first initializes a population of individuals randomly. Then, the fitness of each individual is evaluated. In each iteration, two parents are selected by roulette-wheel selection to perform crossover and mutation, which lead to two offspring. After computing the fitness of each offspring, the population is updated. This is an online-design approach as the hyper-heuristic model is designed when the algorithm is used to solve a problem. More than one combinations of high-level heuristic components are applied throughout the optimization

process. GEP-HH utilizes the automatic generation methodology to generate high-level heuristic components in a similar way as those used to generate LLHs in most of the generation hyper-heuristic methods. While the GP-based rule generation strategies can automatically generate the computer programs, one drawback is the search space size grows exponentially in both the number of components that form a rule and the number of generated rules [11].

In this article, we propose a method to automatically design hyper-heuristics by intelligently selecting the appropriate high-level heuristic components from a set of human-designed alternatives during the optimization process (i.e. online). The proposed method combines the strengths of the two approaches mentioned previously [1,56]. It can lead to a reduction in the search space, as compared with that in Sabar et al. [56], and at the same time, does not require the time-consuming meta-optimization process, as compared with that in Adriaensen et al. [1]. In the proposed method, RL with a state aggregation technique is applied to guide the selection of the high-level heuristic components for different stages of the optimization process. While RL has been commonly applied to various hyper-heuristics, its use is confined to LLH selection in most of the existing studies, as presented in Section 2.2. The novelty of our work is to automatically design the high-level heuristic components (i.e. the LLH selection and move acceptance methods) of the hyper-heuristic model by utilizing Q-learning [67] to guide the selection of these components.

Utilizing Q-learning for automatically designing hyper-heuristics draws certain properties pertaining to a Markov Decision Process (MDP), where the active solutions represent the states and the selectable combinations of high-level heuristic components represent the actions. The automatic design process of hyper-heuristic is initiated by observing the current solution (current state) and selecting a combination of high-level heuristic components (an action). Then, the resulting high-level heuristic is executed starting from the current solution. The execution of the resulting high-level heuristic (selected action) leads the optimization process (environment) into a new state by controlling which LLH to be used in the current solution to produce a new solution, and subsequently deciding whether to accept or reject the new solution. After the execution phase, a reward (or penalty) signal is given to the selected action with respect to the current state based on a reward function.

In an MDP, it is assumed that the states have full observability (i.e. complete information about the states is available). However, in the proposed QHH model, which is aimed to design cross-domain (i.e. SAT, BP, FS, PS, TSP and VRP) hyper-heuristics, the information about the solutions is partially observable. This is because the QHH model only employs solution fitness to model the states. Other information such as internal structure and neighbourhood structures cannot be generalized and employed in the QHH model due to different solution representation schemes of different domains. Therefore, the automatic design process of hyper-heuristic in the proposed QHH model is akin to a Partially Observable MDP (POMDP). Some publications on using RL for solving POMDP include Hausknecht and Stone [25], Jaakkola et al. [30], and Kimura et al. [38].

As RL may suffer from the limitation of slow convergence when the state space is large [62], a number of function approximation techniques for solving large or continuous state spaces problems have been proposed. Hunor and Csató [27] presented a kernel-based value approximation method for on-line RL in continuous state and action spaces. The effectiveness of the kernel-based method was demonstrated using the mountain-car problem in a continuous two-dimensional state space and a continuous action space. Kula and Ocaktan [41] proposed a Fuzzy Semi Markov Average Reward Technique (FuzzySMART), namely an RL algorithm with an adaptive network-based fuzzy inference system for solving semi MDP. The numerical results indicated that FuzzySMART showed fast convergence with respect to a large number of state-action pairs. Jasmin et al. [33] proposed an RL algorithm with neural network-based function approximation for solving the unit commitment problem with photo voltaic sources. The empirical results showed that the proposed neural network-based function approximation technique was useful in handling a large state space related to stochastic renewable energy sources. Powell [54], and Wang et al. [66] presented variations, development, and applications of Approximate Dynamic Programming (ADP) which can be classified into two categories: techniques with initial stable policy, and techniques without the requirement of initial stable policy. Another function approximation technique, namely state aggregation, is commonly used to divide the large state space into several disjoint categories [9]. A few variants of state aggregation techniques have been presented in the literature [5,49,61]. Besides that, Busoniu et al. [9] and Xu et al. [69] provide comprehensive accounts on recent advances and applications of RL with function approximation techniques.

The proposed QHH model aims to design cross-domain (i.e. SAT, BP, FS, PS, TSP and VRP) hyper-heuristics. As these problem domains are NP-hard, and they have large solution search spaces, QHH needs to handle a large number of states. As such, an approximation technique based on the concept of state aggregation [5,49,61] is used to aggregate the state space into several disjoint categories. For each action, the same value is assigned to all states in a specific category. As a result, a disjoint category of states can be seen as one single aggregate state. The details on states representation in QHH are described in Section 4.2.

3. Q-learning

Q-learning is one of the commonly used RL algorithms. In Q-learning, each state-action pair is given a total cumulative reward value (denoted as a Q-value), which is measured by a Q-function (Eq. (1)). Suppose $S=[s_1, s_2, s_3,\ldots,s_n]$ denotes a set of possible states, $A=[a_1, a_2, a_3,\ldots,a_n]$ denotes a set of selectable actions, r_{t+1} denotes the immediate reinforcement signal, $\alpha\in[0,1]$ denotes the learning rate, $\gamma\in[0,1]$ denotes the discount factor, and $Q(s_t, a_t)$ denotes the Q-value at time t, which is calculated using the Q-function, as stated in Eq. (1). The Q-values of all state-action pairs are stored in a Q-table.

$$Q_{t+1}(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma * max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

Table 3
Notations used in QHH.

$Sol_{current}$	The current solution
$f_{current}$	The fitness of $Sol_{current}$
Sol_{new}	The newly produced solution
f_{new}	The fitness of Sol_{new}
Δf	$f_{current}$ - f_{new}
$f_{initial}$	The fitness of the initial solution
t_{max}	The maximum allowed time for algorithm execution
$t_{current}$	The current elapsed time of algorithm execution
D	The duration in *ms* for an LLH to produce $Sol_{new.}$
n	The cumulative number of executions of an LLH from the first to the current iteration
$Avg(x)$	The moving average of x calculated from the first to the current iteration.
$Total(x)$	The accumulative value of x calculated from the first to the current iteration.
$Identical(Sol_{current}, Sol_{new})$	If $Sol_{current} = Sol_{new}$,, return 1; otherwise return 0
$norm(x)$	The normalized value of x.
$s_{initial}$	The initial state.
s_t	The current state

Algorithm 1
Pseudocode of QHH.

```
1   Procedure QHH()
2     Initialization()
      /* initialize Q-table, initial solution, parameters, i.e. NoE (see Section 4.5) and γ (see Eq. (1)) */
3     GlobalBest = f_initial
      /*store the fitness of the initial solution as GlobalBest.
4     Determine_initial_state()
      /* determine s_initial based on norm(f_initial) (see Eq. (2)) */
5       while !stopping _criteria do
6         Design()
          /*select the best action (i.e. an LLH selection–move acceptance pair) of s_t with the largest Q-value */
7         Execution()
          /*execute the designed hyper-heuristic for t_max/NoE */
8         Determine_next_state ()
          /*determine s_{t+1} based on norm(f) of the last accepted solution (see Eq. (2)) */
9         Update_Q-value()
          /*calculate Q(s_t, a_t) based on Eq. (1) */
10          s_t = s_{t+1}
          /*update the state */
11          Update_GlobalBest()
          /*update the value of GlobalBest if a better solution is produced during the last execution */
12      end while
13  end Procedure
```

Q-learning has been extensively adopted in various applications. Mohammad et al. [48] used a Q-learning method for planning the motions of a mobile robot in a dynamic environment. The robot is able to arrive at its target location despite no information pertaining to the environment is known. Wei et al. [68] proposed a "dual iterative Q-learning algorithm" for optimal battery management in smart residential environments. In the proposed model, the external iteration aims to converge the Q-function to the optimum, while the internal iteration is in charge of minimizing the power loads. Mnih et al. [47] applied Q-learning to train a convolutional neural network for playing Atari. The proposed method is able to learn the control policies from high dimensional sensory inputs. The results show the effectiveness of the proposed Q-learning method as compared with a number of existing methods such as SARSA and Contingency in most game engines. In Kiumarsi et al. [39], a Q-learning-based method was proposed to solve the tracking problem of unknown discrete-time linear systems. The proposed method is able to track the control signals without requiring information of the system dynamics. Besides that, Q-learning is applied to unknown continuous-time linear systems in Vamvoudakis [65].

4. The proposed method

In this section, the proposed QHH model, i.e., a Q-learning-based method to automatically design hyper-heuristics, is presented. To design a hyper-heuristic, a high-level heuristic which contains an LLH selection–move acceptance pair needs to be first determined. By adopting Q-learning, QHH is able to intelligently select appropriate LLH selection–move acceptance pairs during different stages of the optimization process.

Before going into the details of QHH, Table 3 shows the notations used in this study. The pseudocode of QHH is as shown in Algorithm 1.

Table 4
The LLH selection methods of QHH.

LLH selection methods	Description
Improvement	max(0, Δf) in the most recently execution.
Accepted	If Sol_{new} is accepted by the move acceptance method, return 1; Otherwise, return 0.
Improvement over time	C^* *Avg*(*Improvement*)/ *Avg*(*D*); *C* is a large constant, i.e. 100,000 [1].
Speed	$(n+1)/Total(D)$.
Speed accepted	$[Total(Accepted)+1]/Total(D)$.
Speed new	$[Total(Accepted$ & not $Identical(Sol_{current}, Sol_{new}))+1]/Total(D)$.

Table 5
The move acceptance methods of QHH.

Move acceptance methods	Description
OI	If $\Delta f > 0$, Sol_{new} replaces $Sol_{current}$.
AM	Sol_{new} always replaces $Sol_{current}$.
NA	If $\Delta f > 0$, Sol_{new} replaces $Sol_{current}$; otherwise, Sol_{new} replaces $Sol_{current}$ with 0.5 probability
SA	Sol_{new} replaces $Sol_{current}$ with $e^{\frac{\Delta f}{T*\mu_{impr}} * \frac{t_{max}}{t_{max}-t_{current}}}$ probability. Note that μ_{impr} is the moving average of improvement in all improved iterations. This method is a variant of SA proposed by Adriaensen et al. [2]. It allows the measurement of Δf to be more instance-dependent. In QHH, T (i.e. the initial temperature) is set to 1.
Accept Probabilistic Worse (APW)	Sol_{new} replaces $Sol_{current}$ with $e^{\Delta f/(T*\mu_{impr})}$ probability. This method is a variant of Metropolis Acceptance proposed by Adriaensen et al. [2]. In QHH, T is set to 0.5.

In QHH, the optimization process is separated into a number of "episodes", as in Eiben et al. [20], RL is applied to control the evolutionary algorithm online. As shown in Algorithm 1, after designing a hyper-heuristic by selecting the best action which represents an LLH selection–move acceptance pair with the largest Q-value (Line 6, Algorithm 1), the designed hyper-heuristic is executed for one episode starting from the last accepted solution of the previous episode (Line 7,Algorithm 1). Then, the next state is identified based on the normalized fitness of the last accepted solution (Line 8, Algorithm 1). The Q-value of the selected action is updated based on the performance of its execution (Line 9, Algorithm 1). After that, the state is updated (Line 10, Algorithm 1) and the current global best solution is recorded (Line 11, Algorithm 1). An action is selected to perform the next episode of optimization (Line 5, Algorithm 1).

4.1. Action representation

To apply Q-learning, a set of actions and states need to be defined. In QHH, an action represents one LLH selection–move acceptance pair. The design of LLH selection and move acceptance alternatives is a modified version of that in Adriaensen et al. [1]. QHH uses six LLH selection methods and five move acceptance methods, resulting in a total of 30 actions. Tables 4 and 5 list the LLH selection and move acceptance methods, respectively.

4.2. State representation

In RL, the state (i.e. s_t) indicates the environmental situation for deciding an action. It characterizes the first solution of the episode (i.e. the last accepted solution from the previous episode). As mentioned in Section 2.3, the state space is divided into several disjoint categories. As the aim is to design cross-domain hyper-heuristics, a fitness-based state categorization that requires less domain-dependent information is used.

Since the fitness of each problem has different ranges, it is normalized by a moving average of fitness (f_{new}) calculated from the first to the current iteration, as in Eq. (2).

$$norm(f) = f/Avg(f_{new}) \tag{2}$$

where $norm(f)$ of the first solution of the episode is categorized into a few intervals to represent different aggregate states. In QHH, there are three aggregate states, i.e., $norm(f) \in [0,0.67)$, $[0.67,1.33)$ and $[1.33,\infty)$.

4.3. Reinforcement signal

In the Q-function (Eq. (1)), there is an RL signal, r, that denotes the reward or penalty. This is a user-defined parameter of Q-learning. By referring to Samma et al. [59], a mechanism to determine r is proposed. The designed hyper-heuristic is executed for one episode. If the global best solution of the last episode can be improved at the end of the current episode, the state-action pair receives a reward of $r = 1$; otherwise, a penalty is imposed, i.e., $r = -1$.

Algorithm 2
The procedure of *Execution* in QHH.

```
1   Procedure Execution()
2     t_start = t_current
3     t_expire = t_start + t_max/NoE
4     while t_current < t_expire do
5       selected_LLH = Select_perturbative_LLH()
        /* select a perturbative LLH based on the chosen LLH selection method */
6       Sol_new = Produce_new_solution(selected_LLH, Sol_current)
        /* Diversification: produce a new solution by using selected_LLH to modify Sol_current */
7       Local_search()
        /* Intensification: perform local search on Sol_new */
8       Move_acceptance()
        /* Decide whether to accept Sol_new based on the chosen move acceptance method */
9       Update_LLH_evaluation_score();
        /* Update various LLH evaluation scores based on the previous performance*/
10    end while
11  end Procedure
```

4.4. Q-function parameters

There are two parameters in the Q-function (Eq. (1)): the discount factor and the learning rate. The discount factor, γ, determines the influence of the future reward. If γ is low, the current reward is emphasized more than the future reward; otherwise, the long-term reward is considered. Rakshit et al. [55] suggested setting γ to 0.8. In QHH, γ is determined by a parameter tuning process, as detailed in Section 5.1.

The learning rate, α, determines the ratio of accepting the newly learned information or keeping the existing information. A high α value indicates a tendency to allow the new information to replace the old information. On the other hand, a low α value encourages more exploitation pertaining to the existing information. In this study, the method in Rakshit et al. [55] to dynamically control this parameter is deployed. As such, a high value of α is used at the initial stage of the search process. It is decreased over time, as in Eq. (3), in order to encourage exploitation in subsequent stages.

$$\boldsymbol{\alpha_t} = 1 - \left(0.9 \times \frac{\boldsymbol{t_{current}}}{\boldsymbol{t_{max}}}\right) \tag{3}$$

4.5. Execution of the designed hyper-heuristic

For evaluation purposes, the designed hyper-heuristic is executed for one episode of the optimization process. An episode of QHH contains a period of time, instead of a number of iterations. This is because the time-based stopping criteria are used in most of the hyper-heuristics which are implemented using HyFlex. If the proposed hyper-heuristic uses the iteration-based stopping criteria, the method of splitting episodes can be changed accordingly.

The *NoE* parameter determines how many equal-period episodes should be generated throughout the t_{max} allowed optimization time. In other words, the period of each episode $\approx t_{max}/NoE$. Parameter *NoE* needs to be set carefully. A high *NoE* indicates a short episode period, with more frequent updates of the Q-table. This can result in inaccurate performance measure of the high-level heuristic. It is especially unfair for those high-level heuristics which contains explorative LLH selection or/and move acceptance methods, because explorative methods usually show slow improvements, but are capable of escaping from the local optimum. On the other hand, a low value *NoE* indicates that a single action is executed for a long period without considering the environmental changes. The design may no longer suit the environment during the long execution period. An appropriate *NoE* setting is able to give correct feedback, and enable the selection of correct actions. Therefore, in QHH, *NoE* is determined with a set of parameter tuning experiments, as presented in Section 5.1. Algorithm 2 shows the pseudocode of the *Execution* procedure (Line 7, Algorithm 1).

As shown in Algorithm 2, the execution of the designed hyper-heuristics follows an ILS procedure, which includes a diversification phase (Line 6, Algorithm 2) and an intensification phase (Line 7, Algorithm 2). In QHH, the diversification phase considers only ruin-recreate heuristics and mutation heuristics, while crossover heuristics are ignored. In Line 5 of Algorithm 2, one LLH (from all the available ruin-recreate heuristics and mutation heuristics) is selected using roulette-wheel selection [24] based on the LLH selection method determined during *Design*. The intensification phase follows a Variable Neighbourhood Descent (VND) procedure [44] that iteratively applies a sequence of all available local search heuristics until no improvement of the current solution. In QHH, the sequence of the available local search heuristics is re-ordered randomly whenever the solution changes. The pseudocode of *Local_search* is shown in Algorithm 3. After the intensification phase, the selected move acceptance method decides whether to replace $Sol_{current}$ with Sol_{new} (Line 8, Algorithm 2). The selected LLH is evaluated with six performance measures corresponding to the six LLH selection methods used in QHH (Line 9, Algorithm 2). Note that all the six scores are updated after the application of a LLH, and not just the score measured by the chosen LLH selection method is updated.

Algorithm 3
Procedure of *Local_search* in QHH.

```
1   Procedure Local_search()
2     Re-order_LS_sequence()
      /* randomly generate a sequence of local search heuristic {LS1, LS2, LS3, ..., LSn} */
3     k = 1
4     while k <= n do
5       Apply_LSk_on_ Solnew()
6       if Solnew is improved do
7         k = 1
8         Re-order_LS_sequence()
9       else
10          k = k + 1
11      end if
12    end while
13  end Procedure
```

Table 6
The available actions of the numerical example.

a_1	Improvement + OI
a_2	Accepted + OI
a_3	Improvement + AM
a_4	Accepted + AM

Table 7
The initial LLH score table of the numerical example.

	Improvement	Accepted
LLH1	MAX	MAX
LLH2	MAX	MAX
LLH3	MAX	MAX

Table 8
The initial Q-table of the numerical example.

	a_1	a_2	a_3
$norm(f) \in [0,0.67)$	0	0	0
$norm(f) \in [0.67,1.33)$	0	0	0
$norm(f) \in [1.33,\infty)$	0	0	0

4.6. Numerical example

A numerical example is provided to clarify the proposed QHH model. Suppose there are only two heuristic selection methods (i.e. *Improvement* and *Accepted*), and two move acceptance method (i.e. OI and AM). This results in a total of four available actions, a_1, a_2, a_3, and a_4, as shown in Table 6. Suppose there are only three perturbative LLHs (mutation or ruin-recreate heuristics), i.e. LLH1, LLH2, and LLH3. Each LLH owns two scores, which correspond to the two LLH selection methods. All the scores of each LLH are set to the maximum value before its first use, as shown in Table 7. This ensures that every LLH is used at least once.

Initially, the Q-values of all state-action pairs are initialized with zero (Table 8). Assume that the fitness of the initial solution (i.e. $f_{initial}$) is 11, and its $norm(f)$ is 1 (i.e. s_t is $norm(f) \in [0.67,1.33)$. The value of *Globalbest* is 11 now (Line 3, Algorithm 1).

Firstly, *Design()* takes place by selecting an action (Line 6, Algorithm 1). Assume that a_1 (i.e. *Improvement* + OI) is selected for this episode. It is executed for t_{max}/NoE period of time. Assume that the execution ends in three iterations. Table 9 shows the details of the execution of this episode.

During *Execution()* (i.e. Line 7, Algorithm 1), an LLH is selected using roulette-wheel selection based on the LLH selection method determined during *Design()*. For this episode, the *Improvement* column in Table 9 is highlighted because *Improvement* is the selected LLH selection method, i.e. the LLH is selected based on the scores in this column. To explain how to update the LLH score table, the first iteration is chosen as an example. In the first iteration, LLH2 is selected (see "selected LLH" column in Table 9). The starting solution whose fitness is 11 (see the $f_{current}$ column in Table 9) is modified by LLH2. After that, the solution goes through a local search procedure, and a solution with fitness of 9 is produced (see the f_{new} column in Table 9). The new solution is accepted by the OI (determined during *Design()*) move acceptance method. The score of LLH2 corresponds to *Improvement* is updated to 2 (i.e. 11–9), while the score corresponds to *Accepted* is updated to 1 (since the solution is accepted).

Table 9
The first episode of the numerical example.

Design: a_1 (*Improvement* + OI), Fitness of the starting solution: 11							
Iteration	$f_{current}$	Selected LLH	f_{new}	Accepted (Yes/No)	Updated LLH score table (the updated row is highlighted)		
					LLH	*Improvement*	*Accepted*
1	11	LLH2	9	Yes	LLH1	MAX	MAX
					LLH2	2	1
					LLH3	MAX	MAX
2	9	LLH3	10	No	LLH1	MAX	MAX
					LLH2	2	1
					LLH3	0	0
3	10	LLH1	5	Yes	LLH1	5	1
					LLH2	2	1
					LLH3	0	0

Table 10
The updated Q-table of the numerical example.

	a_1	a_2	a_3
$norm(f)\in[0,0.67)$	0	0	0
$norm(f)\in[0.67,1.33)$	0.9	0	0
$norm(f)\in[1.33,\infty)$	0	0	0

Table 11
The first iteration of the second episode of the numerical example.

Design: a_2 (*Accepted* + AM), Fitness of the starting solution: 5							
Iteration	$f_{current}$	Selected LLH	f_{new}	Accepted (Yes/No)	Updated LLH score table (the updated row is highlighted)		
					LLH	*Improvement*	*Accepted*
1	5	LLH2	7	Yes	LLH1	5	1
					LLH2	0	1
					LLH3	0	0

The next step is *Determine_next_state()* (i.e. Line 8, Algorithm 1). The next state characterizes the last accepted solution of the first episode. As shown in Table 9, the fitness of the last accepted solution of the first episode is 5. Its fitness is normalized by a moving average of f_{new} calculated from the first to the current iteration, as follows.

$$Avg(f_{new}) = (9 + 10 + 5)/3 = 8$$
$$norm(f) = 5/8 = 0.625 \in [0, 0.67)$$

Therefore, s_{t+1} = "$norm(f)\in[0,0.67)$" .

Then, the Q-table is updated. Assume $\alpha = 0.9$ (α is determined by Eq. (3) in the real implementation) and $\gamma = 0.8$, the Q-value of ($norm(f)\in[0.67,1.33)$, a_1) is (see Eq. (1)):

$$Q_{t+1} = 0 + 0.9\{1 + [0.8max(0, 0, 0)] - 0\} = 0.9.$$

Therefore, r_{t+1} is 1 in this case because the execution of a_1 (i.e. the first episode) provides at least one solution whose fitness < *GlobalBest* (or > if it is a maximization problem). The updated Q-table is shown in Table 10:

Finally, the state is updated, i.e. $s_t = s_{t+1}$ = "$norm(f)\in[0,0.67)$". The value of *GlobalBest* is updated to 5. After that, the next episode starts with *Design()* again. Assume that a_4 (*Accepted* + AM) is selected for this episode. It is executed from the last accepted solution of the previous episode (whose fitness is 5 in this example). Note that the LLH score table is not reinitialized across all episodes. Table 11 shows the first iteration of the second episode.

The process is repeated until t_{max} is reached.

Table 12
Low, medium, and high levels of *NoE* and γ.

	Low	Medium	High
NoE	30	80	130
γ	0.2	0.5	0.8

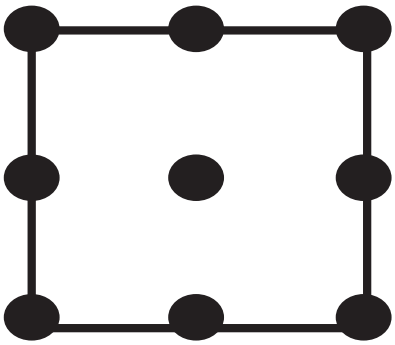

Fig. 3. A face-centred central composite design with two parameters.

5. Experimental studies

This section presents an empirical evaluation of the proposed QHH model. All experiments have been conducted using a computer with multiple Intel i7-3930K 3.20 GHz processors, and with 15.6GB of memory. At any particular time, each test of the experiment has been handled by one processor only. QHH is implemented through HyFlex [50] with NetBeans IDE 5.5 as the development tool. There are three QHH parameters, i.e. *NoE*, γ, and α. Note that α is dynamically controlled (see Section 3.3) while *NoE* and γ are regulated using a set of parameter tuning experiments. The QHH performance is assessed based on thirty instances used in the CHeSC competition [29]. Section 5.1 describes the parameter tuning experiment while Section 5.2 presents the entire experimental results and performance comparison studies.

5.1. Parameter tuning

In QHH, parameter α is dynamically controlled. Two other parameters (i.e. *NoE* and γ) need to be determined via a set of parameter tuning experiments. *NoE* is a positive integer, which indicates the number of episodes splitting the full optimization process, while $\gamma \in [0,1]$ is the discount factor of Q-learning, which controls the future reward of the Q-value. The face-centred central composite design method [4,53] is applied to analyze the effects of *NoE* and γ pertaining to the QHH performance. In accordance with the face-centred central composite design, each parameter is set at three different levels, i.e. low, medium, and high, as shown in Table 12. The experimental setting is illustrated in Fig. 3.

A total of $3^2 = 9$ combinations of these parameters at each level are generated. The comparison uses the Formula One point-scoring system of the CHeSC competition. A total of five replications for each of the nine experiments for each of the 30 CHeSC instances are evaluated. For each replication, t_{max} is determined by a benchmarking software obtained from the CHeSC website [29], to provide a fair comparison for different computing machines. For the machine used for our experimentation, t_{max} is equal to 346 s. The median result is used for comparison. For each of the selected 30 instances, the top performing solution receives 10 points, the second 8 points, and each subsequent solution receives 6, 5, 4, 3, 2, 1 points, respectively. If there are more than eight solutions, all solutions that are worse than 8th solution receive zero point. If there is a tie, the point is equally shared by the solutions. The score of a solution is finally calculated by accumulating the points that it has received over all problems. Table 13 shows the detailed configurations of the face-centred central composite design and the Formula One scores obtained by each parameter setting.

The best result is achieved by the second configuration, i.e. $NoE = 80$ and $\gamma = 0.8$. This configuration has been used in the subsequent experiments and comparison studies. With this configuration, *NoE* is set at the medium level. As mentioned in Section 4.5, a suitable *NoE* value is able to provide an accurate measurement of the performance of high-level heuristics. According to Table 13, the configuration with $NoE = 80$ outperforms all other configurations, regardless of the γ value. As such, 80 is deemed as an appropriate setting for *NoE*. Besides that, the experiments indicate that 0.8 is a suitable value for γ, as suggested by Rakshit et al. [55]. With $NoE = 80$ and 130, the configuration with $\gamma = 0.8$ outperforms other configurations with the same value of *NoE*. Only when $NoE = 30$, $\gamma = 0.8$ show a slight inferior result than that of $\gamma = 0.5$ (i.e. 4.5 points lower).

Table 13
Effects of *NoE* and γ on the QHH performance.

	NoE	γ	Scores
Experiment 1	30	0.5	124.64
Experiment 2	**80**	**0.8**	**183.59**
Experiment 3	130	0.2	111.34
Experiment 4	130	0.8	123.70
Experiment 5	30	0.2	103.39
Experiment 6	80	0.5	151.85
Experiment 7	80	0.2	144.09
Experiment 8	130	0.5	107.24
Experiment 9	30	0.8	120.14

Table 14
The QHH performance for each CHeSC instance over 31 replications.

Problem		Instances				
		1	2	3	4	5
SAT	Best	1	1	0	1	7
	Median	2	3	2	2	8
	Average	2.6	3.4	1.8	2.1	8.4
BP	Best	0.0137	0.0067	0.0101	0.1084	0.0105
	Median	0.0272	0.0075	0.0133	0.1085	0.0168
	Average	0.0276	0.0076	0.0131	0.1085	0.0167
PS	Best	16	9237	3165	1467	325
	Median	24	9862	3260	1688	345
	Average	23.7	9868.4	3260.6	1673.4	345.5
FS	Best	6210	26750	6303	11335	26549
	Median	6244	26790	6345	11389	26592
	Average	6243.7	26798.0	6341.3	11381.3	26586.8
TSP	Best	48194.9	21015638.2	6799.0	66266.3	52319.5
	Median	48194.9	21138157.6	6810.6	66579.8	52899.1
	Average	48202.6	21155330.5	6810.6	66666.8	52973.1
VRP	Best	61035.9	12280.8	144033.0	20653.8	145368.5
	Median	64113.2	13297.8	148253.6	20655.2	146586.0
	Average	65284.4	12892.4	148337.9	20655.7	146752.5

5.2. Experimental results and computational studies

QHH is executed for 31 replications for each of the 30 CHeSC instances. The best, average, and median results for each instance are shown in Table 14. Besides, some additional statistical data such as the maximum point, first quartile, third quartile, interquartile range, and outliers are shown in Fig. 4. Based on the CHeSC rules, the median results in Table 14 are used for calculating the Formula One scores of QHH, which are presented in Table 15.

Besides the 20 CHeSC contestants, the results are compared with eight recent hyper-heuristic models, as follows:

- Automatic design of hyper-heuristic with Gene Expression Programming (GEP-HH) [56].
- GEP-HH with an Adaptive Memory Mechanism (GEP-HH-AMM) [56].
- Fitness Rate Average based Multi-Armed Bandit hyper-heuristic (FRAMAB) [21].
- Automatic generation of move acceptance criteria with GEP for FRAMAB (FRAMAB-GEP) [21].
- Single-point based Harmony Search Memetic Algorithm (SP-MA) [14].
- Population based Harmony Search Memetic Algorithm (POP-MA) [14].
- Fair Share Iterated Local Search (FSILS) [2].
- Population based Monte Carlo Tree Search hyper-heuristic (MCTS-HH) [58].

The first two models (i.e. GEP-HH-AMM and GEP-HH) were proposed by Sabar et al. [56], as described in Section 2.3. FRAMAB and FRAMAB-GEP (A. S. [21]) both use an MAB-based LLH selection method, as described in Table 1. Both SP-MA and POP-MA adopt Harmony Search as the LLH selection method, as described in Table 1. FSILS [2] is automatically designed through an offline meta-optimization process [1] that uses the FocusedILS meta-heuristic as explained in Section 2.3. MCTS-HH uses the MCTS as the LLH selection method, as described in Table 1.

The comparison is conducted based on the CHeSC rules. The median results of the 20 CHeSC contestants can be obtained from the CHeSC website [29]. For the aforementioned eight hyper-heuristic models, the comparison is based on the reported median results in the respective publications. The comparison results are summarised in Table 15.

Based on Table 15, QHH obtains the fifth rank, with a total score of 96.17 among 29 hyper-heuristic models, which is within the 20th percentile. The best performing domain of QHH is the SAT domain. QHH ranks the second in this domain by scoring 29.24 points. It outperforms all other models, except FSILS which scores 37.10 points. Besides that, QHH obtains

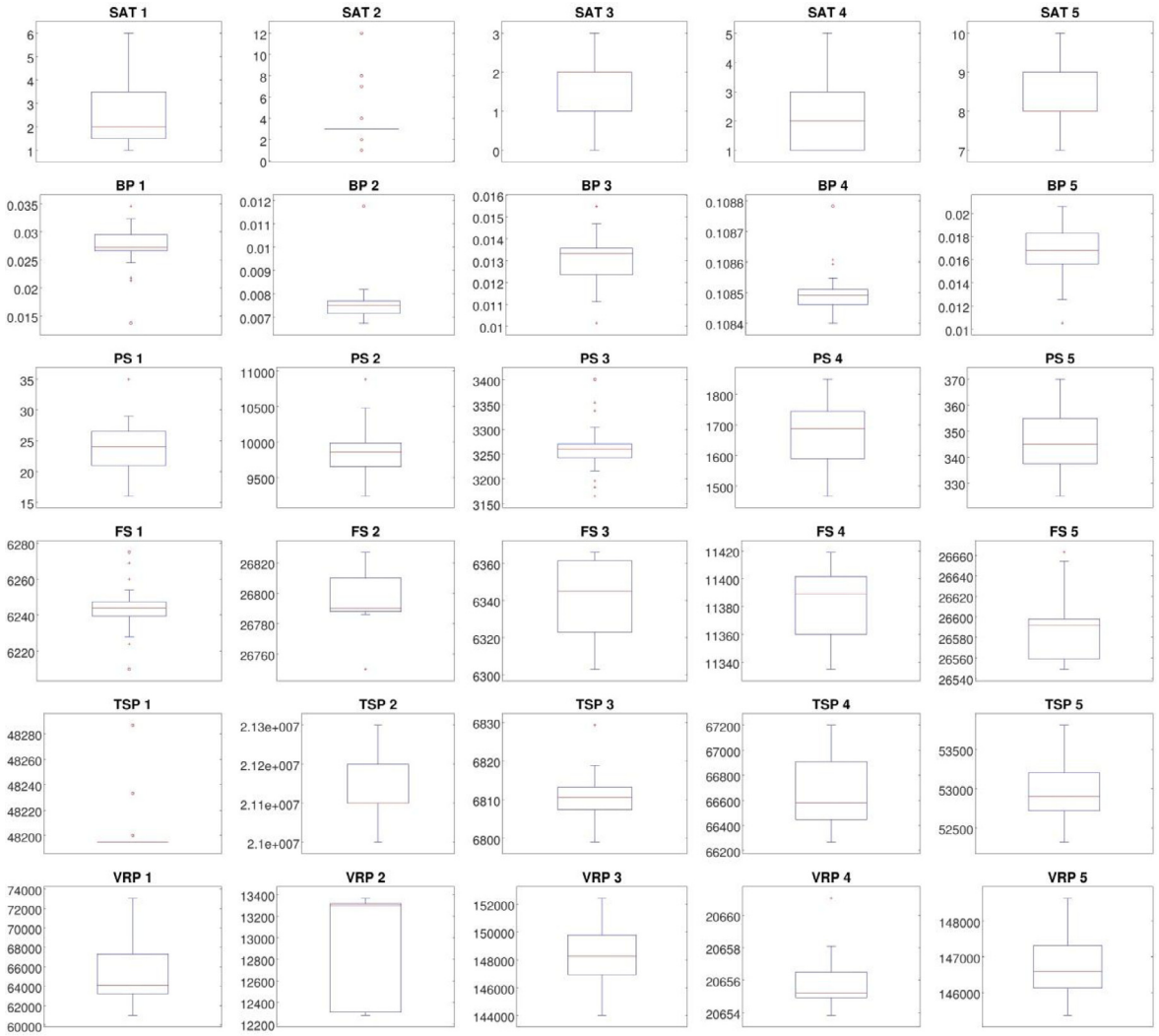

Fig. 4. The box-plots representing QHH performance for each CHeSC instance over 31 replications.

a good score in the TSP domain (i.e. 27.43 points), which is the second best. In the FS domain, QHH scores 24.00 points, which is the third position. In addition, QHH scores 6.50 and 9.00 points, respectively, in the BP and VRP domains. However, QHH does not receive any points in the PS domain.

For ease of comparison, the configurations and characteristics of the top ten approaches in Table 15 are tabulated (as shown in Table 16). The total Formula One scores obtained by the top ten hyper-heuristic models in Table 15 are plotted as a bar chart in Fig. 5.

It is interesting that QHH scores better than GEP-HH, which is also an online-designed hyper-heuristic model. However, by including an adaptive memory mechanism, GEP-HH-AMM is able to obtain the first rank out of all 29 hyper-heuristic models. This indicates that the adaptive memory mechanism, which maintains a set of fit and diverse solutions, is able to strike a good balance between intensification and diversification. Another hyper-heuristic model that utilizes a memory mechanism is MCTS-HH, which is in the second position. The success of GEP-HH-AMM and MCTS-HH indicates the effectiveness of the multi-point search methodology. The hyper-heuristic model in the third position (i.e. FSILS [1]) is designed using a long period meta-optimization process. The training instances used during the meta-optimization process are exact the 30 CHeSC instances. The winner of CHeSC 2011 (i.e. AdapHH) scores the fourth place. Although AdapHH is still competitive, this implies that after CHeSC 2011, research in hyper-heuristic moves rapidly, since there are three methods that outperform AdapHH. AdapHH applies a probabilistic relay hybridization technique to determine the consecutive heuristic. This differentiates it from the traditional selection hyper-heuristic models which usually consider only a single LLH during the LLH selection stage. MCTS-HH (the second position) has the same properties for considering a sequence of LLHs. In MCTS-HH, a sequence of LLHs is represented with a tree, and the LLHs in the tree are executed to produce a solution in a specific order.

Table 15
Ranking of QHH with respect to the 20 CHeSC contestants and eight recent hyper-heuristic models.

Rank	Model	Scores for each domain						Total score
		SAT	BP	PS	FS	TSP	VRP	
1	GEP-HH-AMM	25.24	29.00	23.10	10.50	15.43	21.00	124.27
2	MCTS-HH	17.74	30.00	28.93	9.50	17.43	18.00	121.60
3	FSILS	37.10	1.00	0.00	34.00	30.43	18.00	120.53
4	AdapHH	16.14	42.00	1.00	23.00	23.25	5.00	110.39
5	**QHH**	**29.24**	**6.50**	**0.00**	**24.00**	**27.43**	**9.00**	**96.17**
6	GEP-HH	7.00	23.50	12.50	25.50	10.43	10.00	88.93
7	ML	1.50	3.00	22.60	20.50	4.00	12.00	63.60
8	VNS-TW	19.74	0.00	25.33	17.00	1.25	0.00	63.33
9	FRAMAB-GEP	1.50	0.00	17.93	16.50	5.43	10.00	51.36
10	FRAMAB	1.50	1.00	20.00	0.00	0.00	22.00	44.50
11	NAHH	8.00	13.00	0.00	12.50	6.00	3.00	42.50
12	PHUNTER	1.50	0.00	4.00	0.00	8.25	21.00	34.75
13	ISEA	0.00	19.00	10.10	0.00	5.00	0.00	34.10
14	EPH	0.00	1.00	1.50	0.00	16.25	9.00	27.75
15	POP-MA	0.00	0.00	1.50	2.00	16.43	5.00	24.93
16	HAHA	13.64	0.00	9.00	0.00	0.00	2.00	24.64
17	KSATS-HH	7.14	4.00	0.00	0.00	0.00	7.00	18.14
18	HAEA	0.00	0.00	0.00	0.00	4.00	13.00	17.00
19	ACO–HH	0.00	14.00	0.00	0.00	1.00	0.00	15.00
20	AVEG-Nep	8.00	0.00	0.00	0.00	0.00	4.00	12.00
21	GISS	0.00	0.00	8.00	0.00	0.00	3.00	11.00
22	SA-ILS	0.00	0.00	8.50	0.00	0.00	0.00	8.50
23	XCJ	0.00	5.00	0.00	0.00	0.00	1.00	6.00
24	DynILS	0.00	2.00	0.00	0.00	2.00	0.00	4.00
25	GenHive	0.00	1.00	1.00	0.00	0.00	2.00	4.00
26	SelfSearch	0.00	0.00	0.00	0.00	1.00	0.00	1.00
27	MCHH-S	0.00	0.00	0.00	0.00	0.00	0.00	0.00
28	Ant-Q	0.00	0.00	0.00	0.00	0.00	0.00	0.00
29	SP-MA	0.00	0.00	0.00	0.00	0.00	0.00	0.00

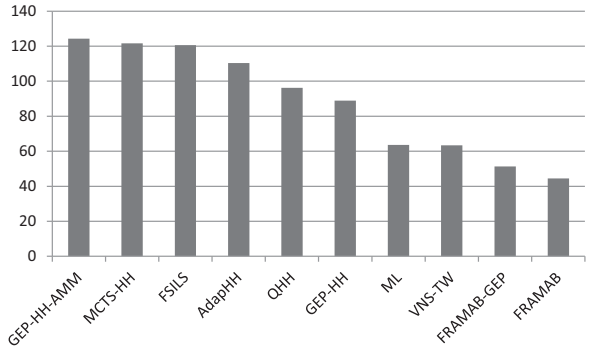

Fig. 5. The total Formula One scores of the top ten models in Table 15.

ML, VNS-TW, FRAMAB-GEP, and FRAMAB are manually designed models which use a multi-point search methodology. They are with the position of 7th, 8th, 9th, and 10th, respectively. The proposed QHH model outperforms these four models. Out of the eight recent hyper-heuristic models, SP-MA and POP-MA are not ranked within the top ten positions.

Besides conducting the comparison using the Formula One scoring system, the Wilcoxon signed rank test is conducted for comparing QHH with each of the hyper-heuristic models within the top ten ranking (as shown in Table 15). In the Wilcoxon signed rank test, the median results of all the problem instances are normalized between zero and one such that all problems are treated with the same priority. The difference between the normalized median results of two hyper-heuristics is ranked. The sum of ranks for the instances in which QHH outperforms its competitor is denoted as R^+, while R^- denotes the sum of ranks for the instances in which QHH is inferior to its competitor. The results of the Wilcoxon signed ranks test are summarized in Table 17.

Table 17 indicates that, based on the 95% confidence interval, QHH is comparable with seven out of the top eight hyper-heuristic models (i.e. p-value $> .05$), and it outperforms FRAMAB and FRAMAB-GEP significantly, with p-value $< .05$ and $R^+ > R^-$.

Table 16
The configurations and characteristics of the top ten hyper-heuristic models in Table 15.

Rank	Model	Citation	Description	Design			Multi-point	Single-point
				manual	Automatic			
					online	offline		
1	GEP-HH-AMM	[56]	The model automatically designs hyper-heuristics by using GEP (Section 2.3). It has an embedded adaptive memory mechanism that stores a set of good and diverse solutions.		√		√	
2	MCTS-HH	[58]	The model uses the MCTS LLH selection method (Table 1).	√			√	
3	FSILS	[2]	The model is designed through an offline meta-optimization process using FocusedILS (Section 2.3).			√		√
4	AdapHH	[46]	This is the winning model of CHeSC 2011. It consists of a few features, i.e. an adaptive dynamic heuristic set, relay hybridization, and an adaptive iteration limited list-based threshold acceptance.	√				√
5	**QHH**	Presented in this article	This proposed model automatically designs hyper-heuristics by intelligently selecting appropriate high-level heuristic components using Q-learning.		√			√
6	GEP-HH	[56]	The model automatically designs hyper-heuristics by using GEP (Section 2.3).		√			√
7	ML	[42]	The model won the third place in CHeSC 2011. It is designed based on the Agent Meta-heuristic Framework [45]. LLHs are selected based on RL and mimetism learning.	√			√	
8	VNS-TW	[26]	The model won the second place in CHeSC 2011. It follows a variable neighbourhood search model which consists of four steps, i.e. shaking, local search, environmental selection, and periodical adjustment.	√			√	
9	FRAMAB-GEP	[21]	The model is a variant that uses the MAB LLH selection method (Table 1). The move acceptance is generated by GEP.	√			√	
10	FRAMAB	[21]	The model is a variant that uses the MAB LLH selection method (Table 1). The move acceptance is SA (Table 2).	√			√	

Table 17
The Wilcoxon signed rank test results between QHH and nine top-performing hyper-heuristic models.

Comparison (QHH vs ...)	R^+	R^-	p-value
GEP-HH-AMM	166	212	.58232
MCTS-HH	152	199	.5485
FS-ILS	132	219	.27134
AdapHH	212	139	.35238
GEP-HH	243	135	.1936
ML	161	190	.71138
VNS-TW	208	170	.64552
FRAMAB	319	116	.02852
FRAMAB-GEP	292	114	.04236

6. Conclusions

A hyper-heuristic is an automated methodology for selecting or generating heuristics. The design of the high-level heuristic of a hyper-heuristic model is crucial, and is problem-dependent owing to the relevant landscape structures of different problems. This paper presents a method to automatically design a hyper-heuristic model by intelligently selecting appropriate combinations of the high-level heuristic components (i.e. LLH selection-move acceptance pairs) during different stages of the optimization process. An RL technique i.e. Q-learning, is applied such that the proposed QHH model is able to learn from experience by interacting with the environment.

In the proposed QHH model, there are a total of 30 actions, which are formed by the combination of six LLH selection methods and five move acceptance methods. A fitness-based state representation and a reinforcement scheme are formulated. The optimization process is divided into multiple episodes. Each episode contains a period of execution time. After designing the hyper-heuristic model, it is executed for an episode before its evaluation takes place (i.e. update the Q-value). The execution follows an iterated local search procedure, in which each iteration consists of a diversification phase and an intensification phase. The QHH parameters are determined through a face centred central composite design method.

The proposed QHH model has been evaluated using thirty benchmark instances from six problem domains in HyFlex. Using the Formula One score system, QHH achieves the fifth position among 29 hyper-heuristic models, which is within the 20th percentile. The Wilcoxon signed rank test also indicates that QHH is competitive with those top-performing hyper-heuristic models.

For future work, the QHH model can be integrated with multi-point search strategy. In the current setting, the proposed QHH model utilizes a single-point search strategy, i.e. it considers a single solution at a particular iteration. If a multi-point search strategy is employed, a pool of solutions can be maintained such that the proposed QHH model is able to select one solution from the pool as the incumbent solution. This multi-point strategy may enhance the algorithm ability in dealing with complex search space (i.e. huge and heavily constrained).

Acknowledgement

The authors gratefully acknowledge the support of the Research University Grant (Grant No: 1001/PKOMP/814274) of Universiti Sains Malaysia for this research. Also, the first author acknowledges the Ministry of Higher Education of Malaysia for the MyPhD scholarship to study for the Ph.D. degree at the Universiti Sains Malaysia (USM).

References

[1] S. Adriaensen, T. Brys, A. Nowe, Designing reusable metaheuristic methods: a semi-automated approach, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), 2014, pp. 2969–2976.
[2] S. Adriaensen, T. Brys, A. Nowe, Fair-share ILS: a simple state-of-the-art iterated local search hyperheuristic, in: Proceedings of the Conference on Genetic and Evolutionary Computation, ACM, 2014, pp. 1303–1310.
[3] S. Asta, Machine learning for improving heuristic optimisation Ph.D. thesis, University of Nottingham, 2015.
[4] M. Balachandran, S. Devanathan, R. Muraleekrishnan, S.S. Bhagawan, Optimizing properties of nanoclay–nitrile rubber (NBR) composites using face centred central composite design, Mater. Des. 35 (2012) 854–862.
[5] J. Baras, V. Borkar, A learning algorithm for Markov decision processes with adaptive state aggregation, in: Proceedings of the Thirty-Ninth IEEE Conference on Decision and Control, 4, IEEE, 2000, pp. 3351–3356.
[6] E.K. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, J.A. Vázquez-Rodríguez, M. Gendreau, Iterated local search vs hyper-heuristics: Towards general-purpose search algorithms, in: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, 2010, pp. 1–8.
[7] 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.
[8] E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, J.R. Woodward, A classification of hyper-heuristic approaches, in: Handbook of Metaheuristics, Springer, 2010, pp. 449–468.
[9] L. Busoniu, R. Babuska, B. De Schutter, D. Ernst, Reinforcement Learning and Dynamic Programming Using Function Approximators, 39, CRC press, 2010.
[10] K. Chakhlevitch, P. Cowling, Hyperheuristics: Recent developments, in: Adaptive and Multilevel Metaheuristics, Springer-Verlag, 2008, pp. 3–29.
[11] L. Chen, H. Zheng, D. Zheng, D. Li, An ant colony optimization-based hyper-heuristic with genetic programming approach for a hybrid flow shop scheduling problem, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), IEEE, 2015, pp. 814–821.
[12] S.S. Choong, L.P. Wong, C.P. Lim, An artificial bee colony algorithm with a modified choice function for the traveling salesman problem, in: Proceedings of the 2017 IEEE International Conference on Systems, Man, and Cybernetics(SMC), 2017, pp. 357–362.
[13] P. Cowling, G. Kendall, E. Soubeiga, A hyperheuristic approach to scheduling a sales summit, in: Practice and Theory of Automated Timetabling III, Springer, 2000, pp. 176–190.
[14] P. Dempster, J.H. Drake, Two frameworks for cross-domain heuristic and parameter selection using harmony search, in: Harmony Search Algorithm, Springer, 2016, pp. 83–94.
[15] E.V. Denardo, Dynamic Programming: Models and Applications, Courier Corporation, 2012.
[16] J. Denzinger, M. Fuchs, M. Fuchs, High Performance ATP Systems By Combining Several AI Methods, University of Kaiserslautern, 1996 Technical Report, SEKI-Report SR-96-09.
[17] L. Di Gaspero, T. Urli, Evaluation of a family of reinforcement learning cross-domain optimization heuristics, in: Learning and Intelligent Optimization, Springer, 2012, pp. 384–389.
[18] J.H. Drake, E. Ozcan, E.K. Burke, A modified choice function Hyper-heuristic controlling unary and binary operators, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2015), 2015.
[19] J.H. Drake, E. Özcan, E.K. Burke, An improved choice function heuristic selection for cross domain heuristic search, in: Proceedings of the Conference on Parallel Problem Solving from Nature-PPSN XII, Springer, 2012, pp. 307–316.
[20] A.E. Eiben, M. Horvath, W. Kowalczyk, M.C. Schut, Reinforcement learning for online control of evolutionary algorithms, in: Proceedings of the International Workshop on Engineering Self-Organising Applications, Springer, 2006, pp. 151–160.
[21] A.S. Ferreira, A cross-domain multi-armed bandit hyper-heuristic Master thesis, Federal University of Parana, 2016.

[22] A.S. Ferreira, R.A. Gon, A.T.R. Pozo, A multi-armed bandit hyper-heuristic, in: Proceedings of Brazilian Conference on Intelligent Systems (BRACIS), IEEE, 2015, pp. 13–18.
[23] C. Ferreira, Gene expression programming: Mathematical modeling by an artificial intelligence, 21, Springer, New York, USA, 2006.
[24] D.E. Goldberg, K. Deb, A comparative analysis of selection schemes used in genetic algorithms, in: Proceedings of the Foundations of Genetic Algorithms, 1991, pp. 69–93.
[25] M. Hausknecht, P. Stone, Deep Recurrent Q-learning for partially observable MDPs, in: Proceedings of the AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents, 2015.
[26] P. Hsiao, T. Chiang, L. Fu, A variable neighborhood search-based hyperheuristic for cross-domain optimization problems in CHeSC 2011 competition, in: Proceedings of the Fifty-Third Conference of OR Society (OR53), Nottingham, UK, 2011.
[27] S.J. Hunor, L. Csató, Novel feature selection and kernel-based value approximation method for reinforcement learning, in: Proceedings of the International Conference on Artificial Neural Networks, Springer, 2013, pp. 170–177.
[28] F. Hutter, H.H. Hoos, T. Stützle, Automatic algorithm configuration based on local search, in: Proceedings of the Twenty-Second National Conference on Artificial intelligence, 2, AAAI Press, 2007, pp. 1152–1157.
[29] M. Hyde, G. Ochoa, The cross-domain heuristic search challenge (CHeSC 2011), 2011. http://www.asap.cs.nott.ac.uk/chesc2011/.
[30] T. Jaakkola, S.P. Singh, M.I. Jordan, Reinforcement learning algorithm for partially observable Markov decision problems, in: Proceedings of the Advances in Neural Information Processing Systems, 1995, pp. 345–352.
[31] W.G. Jackson, E. Ozcan, J.H. Drake, Late acceptance-based selection hyper-heuristics for cross-domain heuristic search, in: Proceedings of the Thirteenth UK Workshop on Computational Intelligence (UKCI), IEEE, 2013, pp. 228–235.
[32] W.G. Jackson, E. Özcan, R.I. John, Fuzzy adaptive parameter control of a late acceptance hyper-heuristic, in: Proceedings of the Fourteenth UK Workshop on Computational Intelligence (UKCI), IEEE, 2014, pp. 1–8.
[33] E.A. Jasmin, T.I. Ahamed, T. Remani, A function approximation approach to reinforcement learning for solving unit commitment problem with photo voltaic sources, in: Proceedings of the IEEE International Conference on Power Electronics, Drives and Energy Systems (PEDES), IEEE, 2016, pp. 1–6.
[34] José Luis Núñez, A.C. (2011) *A general purpose hyper-heuristic based on ant colony optimization* http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/nunez-chesc.pdf.
[35] M. Kalender, A. Kheiri, E. Özcan, E.K. Burke, A greedy gradient-simulated annealing selection hyper-heuristic, Soft Comput. 17 (12) (2013) 2279–2292.
[36] I. Khamassi, Ant-Q hyper heuristic approach applied to the cross domain heuristic search challenge problems, 2011. http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/khamassi-chesc.pdf.
[37] A. Kheiri, E. Keedwell, A sequence-based selection hyper-heuristic utilising a hidden markov model, in: Proceedings of the Annual Conference on Genetic and Evolutionary Computation, ACM, 2015, pp. 417–424.
[38] H. Kimura, K. Miyazaki, S. Kobayashi, Reinforcement learning in POMDPs with function approximation, in: Proceedings of the Fourteenth International Conference on Machine Learning, 97, Morgan Kaufmann Publishers Inc, 1997, pp. 152–160. 152–160.
[39] B. Kiumarsi, F.L. Lewis, H. Modares, A. Karimpour, M. Naghibi-Sistani, Reinforcement Q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics, Automatica 50 (4) (2014) 1167–1175.
[40] G.K. Koulinas, K.P. Anagnostopoulos, A new tabu search-based hyper-heuristic algorithm for solving construction leveling problems with limited resource availabilities, Autom. Constr. 31 (2013) 169–175.
[41] U. Kula, B. Ocaktan, A reinforcement learning algorithm with fuzzy approximation for semi Markov decision problems, J. Intell. Fuzzy Syst. 28 (4) (2015) 1733–1744.
[42] M. Larose, A hyper-heuristic for the CHeSC 2011, in: Proceedings of the first Cross-domain Heuristic Search Challenge, CHeSC Competition 2011, School Computer Science, University of Nottingham, 2011.
[43] H.R. Lourenço, O.C. Martin, T. Stützle, Iterated local search: Framework and applications, in: Handbook of Metaheuristics, Springer, 2010, pp. 363–397.
[44] R. M'hallah, An iterated local search variable neighborhood descent hybrid heuristic for the total earliness tardiness permutation flow shop, Int. J. Prod. Res. 52 (13) (2014) 3802–3819.
[45] D. Meignan, J.-C. Créput, A. Koukam, An organizational view of metaheuristics, in: Proceedings of the First International Workshop on Optimisation in Multi-Agent Systems, AAMAS, 8, 2008, pp. 77–85.
[46] M. Misir, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, A new hyper-heuristic implementation in HyFlex: a study on generality, in: Proceedings of the Fifth Multidisciplinary International Scheduling Conference: Theory & Application, 2011, pp. 374–393.
[47] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, L. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, Technical Report, 2013. Deep Technologies. arXiv:1312.5602.
[48] A.K.J. Mohammad, A.R. Mohammad, Q. Lara, Reinforcement based mobile robot navigation in dynamic environment, Robot. Comput. Integrated Manuf. 27 (1) (2011) 135–149.
[49] T. Mori, S. Ishii, Incremental state aggregation for value function estimation in reinforcement learning, IEEE Trans. Syst. Man Cybern Part B (Cybernetics) 41 (5) (2011) 1407–1416.
[50] G. Ochoa, M. Hyde, T. Curtois, J.A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. Mccollum, A.J. Parkes, S. Petrovic, Hyflex: A benchmark framework for cross-domain heuristic search, in: Evolutionary Computation in Combinatorial Optimization, Springer, 2012, pp. 136–147.
[51] E. Özcan, B. Bilgin, E.E. Korkmaz, A comprehensive analysis of hyper-heuristics, Intell. Data Anal. 12 (1) (2008) 3–23.
[52] E. Özcan, M. Mısır, G. Ochoa, E.K. Burke, A reinforcement learning: Great-deluge hyper-heuristic, Int. J. Appl. Metaheuristic Comput. (IJAMC) 1 (1) (2012) 39–59.
[53] R.M. Pabari, Z. Ramtoola, Application of face centred central composite design to optimise compression force and tablet diameter for the formulation of mechanically strong and fast disintegrating orodispersible tablets, Int. J. Pharm. 430 (1) (2012) 18–25.
[54] W.B. Powell, Approximate Dynamic Programming: Solving the Curses of Dimensionality, 703, John Wiley & Sons, New York, 2007.
[55] P. Rakshit, A. Konar, P. Bhowmik, I. Goswami, S. Das, L.C. Jain, A.K. Nagar, Realization of an adaptive memetic algorithm using differential evolution and Q-learning: A case study in multirobot path planning, IEEE Trans. Syst. Man Cybern. Syst. 43 (4) (2013) 814–831.
[56] N.R. Sabar, M. Ayob, G. Kendall, R. Qu, Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems, IEEE Trans. Evolut. Comput. 19 (3) (2015) 309–325.
[57] N.R. Sabar, M. Ayob, G. Kendall, R. Qu, A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems, IEEE Trans. Cybern. 45 (2) (2015) 217–228.
[58] N.R. Sabar, G. Kendall, Population based monte carlo tree search hyper-heuristic for combinatorial optimization problems, Inf. Sci. 314 (2015) 225–239.
[59] H. Samma, C.P. Lim, J.M. Saleh, A new reinforcement learning-based memetic particle swarm optimizer, Appl. Soft Comput. 43 (2016) 276–297.
[60] K. Sim, KSATS-HH: a simulated annealing hyper-heuristic with reinforcement learning and tabu-search, 2011. http://www.asap.cs.nott.ac.uk/external/chesc2011/index.html.
[61] S.P. Singh, T. Jaakkola, M.I. Jordan, Reinforcement learning with soft state aggregation, in: Proceedings of the Advances in Neural Information Processing Systems, 1995, pp. 361–368.
[62] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.
[63] R.S. Sutton, A.G. Barto, Sarsa: On-policy TD control, in: Reinforcement Learning: An Introduction, MIT Press, 1998, pp. 134–136.
[64] E.G. Talbi, Metaheuristics: From design to implementation, 74, John Wiley & Sons, 2009.
[65] K.G. Vamvoudakis, Non-zero sum nash Q-learning for unknown deterministic continuous-time linear systems, Automatica 61 (2015) 274–281.
[66] F.-Y. Wang, H. Zhang, D. Liu, Adaptive dynamic programming: An introduction, IEEE Comput. Intell. Mag. 4 (2) (2009).
[67] C.J.C.H. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3) (1992) 279–292.

[68] Q. Wei, D. Liu, G. Shi, A novel dual iterative-learning method for optimal battery management in smart residential environments, IEEE Trans. Ind. Electron. 62 (4) (2015) 2509–2518.
[69] X. Xu, L. Zuo, Z. Huang, Reinforcement learning algorithms with function approximation: Recent advances and applications, Inf. Sci. 261 (2014) 1–31.
[70] F. Xue, C. Chan, W. Ip, C. Cheung, Pearl hunter: A hyper-heuristic that compiles iterated local search algorithms, School of Computer Science, University of Nottingham, 2011 Technical Report.
[71] K.Z. Zamli, B.Y. Alkazemi, G. Kendall, A tabu search hyper-heuristic strategy for t-way test suite generation, Appl. Soft Comput. 44 (2016) 57–74.