DOI: 10.1016/j.ins.2018.01.005

Please cite this article as:

S. S. Choong, L. P. Wong, and C. P. Lim, "Automatic design of hyper-heuristic based on reinforcement learning," Information Sciences, vol. 436-437, pp. 89–107, 2018.

Automatic Design of Hyper-heuristic Based on Reinforcement Learning

[1] Shin Siang Choong, [2] Li-Pei Wong, [3] Chee Peng Lim

[1,2] School of Computer Sciences, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia

[3] Institute for Intelligent Systems Research and Innovation, Deakin University, Australia

{[1] css15_com047@student.usm.my, [2] lpwong@usm.my, [3] chee.lim@deakin.edu.au}

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.

Keywords: hyper-heuristic; Q-learning; automatic design; cross-domain heuristic search

I. 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 (Talbi, 2009). 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 (Burke, et al., 2013). The term "hyper-heuristic" was coined by Denzinger, et al. (1996). Figure I shows a classification of hyper-heuristic approaches. There are two main hyper-heuristic categories, i.e. selection hyper-heuristic and generation hyper-heuristic (Burke, Hyde, et al., 2010). These two categories can be defined as 'heuristics to select heuristics' and 'heuristics to generate heuristics' respectively (Burke, et al., 2013). The heuristics to be selected or generated in a hyper-heuristic model are known as the low-level heuristics (LLHs). Both selection and generation hyper-heuristics can be further divided into two categories based on the nature of the LLHs (Burke, Hyde, et al., 2010), 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. (2012), 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.

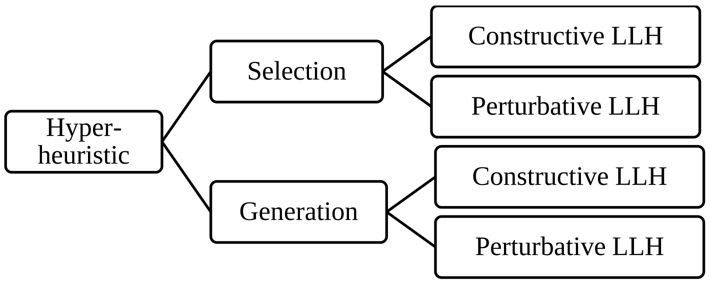

Figure I: A classification of hyper-heuristic models.

A traditional selection hyper-heuristic model consists of two levels, as shown in Figure II (Burke, et al., 2013). 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 (Özcan, et al., 2008). 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 (Sabar, et al., 2015a). 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.

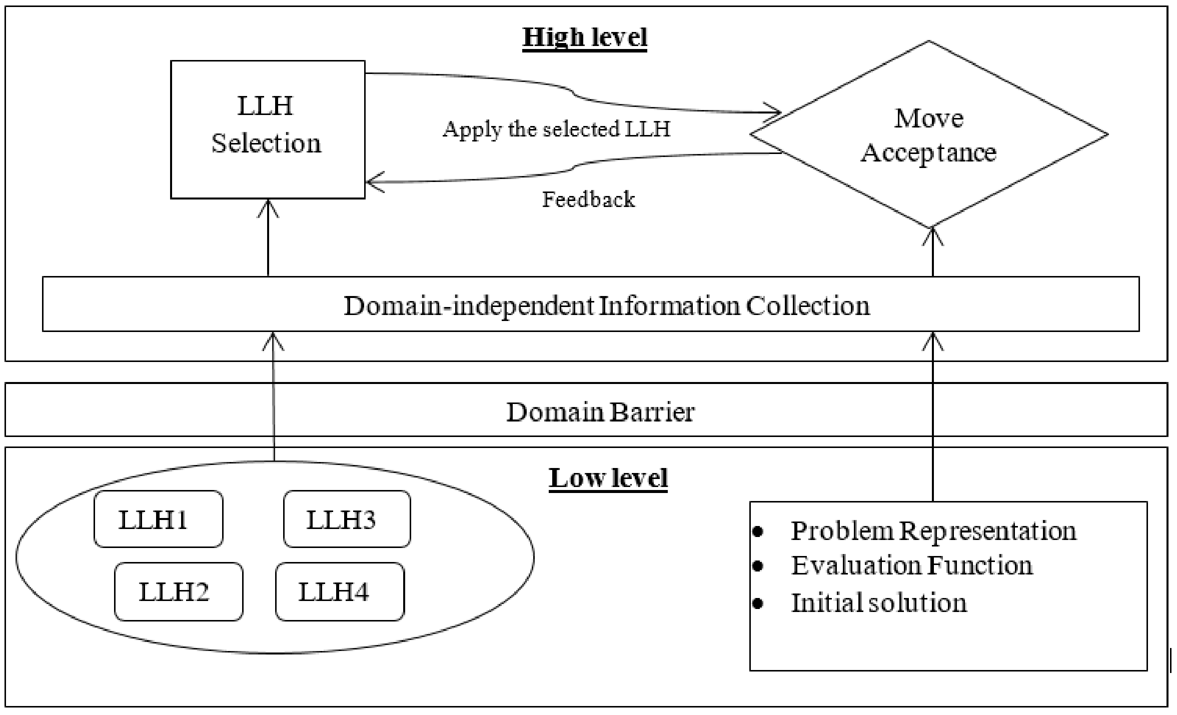

Figure II: A generic structure of a traditional hyper-heuristic model.

There are many different pairwise combinations of LLH selection and move acceptance methods in the literature (Burke, et al., 2013). However, in most of the existing hyper-heuristic models, the high-level heuristic is designed manually (Sabar, et al., 2015a). 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) (Sutton & Barto, 1998a) 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) (Denardo, 2012), SARSA (Sutton & Barto, 1998b), and Q-learning (Watkins & Dayan, 1992).

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 (Ochoa, et al., 2012) is used, and the results indicate the competitiveness of QHH.

The remaining of this article contains a description of related work in Section **Error! Reference source not found.**, 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.

II. Related Work

In this section, a review on hyper-heuristics is presented in Section II.A. Section II.B highlights the applications of RL to hyper-heuristics, while Section II.C describes a number of methods to automatically design hyper-heuristics.

A. Hyper-heuristics

TSP is one of the combinatorial optimization problems that has attracted great research attention. This section presents a brief description about TSP and some related studies about the application of the ABC algorithm to TSP. The neighbourhood search heuristics used by these algorithms are highlighted.

TSP is an NP-hard discrete combinatorial optimization problem [27]. When solving a TSP, the aim is to look for the shortest Hamiltonian path, which is the route that leads a person to visit each location once and only once and to return to the starting location with the minimum total distance [28]. Suppose that the cities are located in some geometric region that the distances between two cities obey the usual axioms of a distance function for a metric space. TSP can be modelled as an undirected weighted graph. Let $G = (V, E)$ be a directed or undirected weighted complete graph, where V is a set of n cities ($V = \{v_1, v_2, . . . , v_n\}$) and E is a set of edges ($E = \{(r, s) : r, s \in V\}$). E is usually associated with a distance matrix, $D = \{d_{r,s}\}$ where $d_{r,s}$ refers to the distance between city r and city s. Let $\prod$ represents all possible permutations of set V. A solution of a TSP is to determine a permutation $\pi \in \prod$, which has the minimum total round trip distance, as shown in Eq. (2), where $\pi(i) \in V$ indicates the i-th element in π.

$$C_{TSP}(\pi \in \prod) = \sum_{i=1}^{n-1}[d_{\pi(i),\pi(i+1)}] + d_{\pi(n),\pi(1)} \tag{2}$$

Banharnsakun, et al. [29] extended the ABC algorithm with a Greedy Subtour Crossover (GSX) heuristic [30] to solve TSP, which is denoted as ABC-GSX. Specifically, GSX is adopted as the neighbourhood search heuristic. In ABC-GSX, the new solutions generated during the neighbourhood search are further improved by using the 2-opt local search heuristic. Banharnsakun, et al. [29] stated that GSX improves the exploitation process of the ABC algorithm. Karaboga and Gorkemli [31] proposed a combinatorial ABC algorithm to solve TSP. A Greedy Sub-tour Mutation (GSTM) heuristic serves as the neighbourhood search heuristic of the employed and onlooker bees. The resulting algorithm is denoted as ABC-GSTM. The GSTM heuristic was proposed by Albayrak and Allahverdi [32] as an operator in the Genetic Algorithm (GA). In the experiments of Karaboga and Gorkemli [31], ABC-GSTM outperforms eight GA variants with different mutation operators. Li, et al. [33] applied an inner-over operator [34] as the neighbourhood search heuristic in ABC. The inner-over operator is a modified version of the inversion mutation. However, the selection of a sub-sequence to be inverted is related to the population, therefore the operator has some features of the crossover heuristic. The ABC algorithm with inner-over operator outperforms another bee-inspired model, namely Bee Colony Optimization [35]. Kıran, et al. [36] analyzed the effect of integrating single and multiple neighbourhood search heuristic(s) in a discrete ABC model. The heuristics include Random Swap (RS), Random Insertion (RI), Random Swap of Subsequences (RSS), Random Insertion of Subsequence (RIS), Random Reversing of Subsequence (RRS), Random Reversing Swap of Subsequences (RRSS), and Random Reversing Insertion of Subsequence (RRIS). The experiments conducted in [33] can be divided into two categories. The first category consists of seven ABC models which employs a single neighbourhood search heuristic. The second category consists of two ABC models which employ multiple neighbourhood search heuristics (i.e. [RS, RSS, RRSS] and [RI, RIS, RRIS]). When multiple neighbourhood search heuristics are employed, a random selection strategy is applied. The empirical results show that the model with multiple neighbourhood search heuristics (i.e. [RI, RIS, RRIS]) has a better performance on the TSP instances with the number of cities ranging between 30 and 101. The model with RRS as the only neighbourhood search heuristic performs better in two TSP instances with 225 and 280 cities, respectively.

Based on the literatures reviewed in this section, the ABC model can be integrated with a single or multiple neighbourhood search heuristic(s). This article proposes a new ABC model with multiple neighbourhood search heuristics. Specifically, we use the MCF to guide the selection of the neighbourhood search heuristics (i.e. LLHs) in the proposed ABC-MCF model.

B. Modified Choice Function

Cowling, et al. [13] proposed a hyper-heuristic based on a choice function. It is a score-based approach which measures the score of each LLH based on its previous performance. The score of each LLH is composed by three different measurements, i.e. f_1, f_2, and f_3. The first measurement, f_1, represents the recent performance of each LLH (see Eq. (3)) where h_j denotes the j-th LLH, $I_n(h_j)$ denotes the fitness difference between the newly proposed solution by h_j and the current solution, $T_n(h_j)$ denotes the amount of time taken to propose the new solution, $\alpha \in (0,1)$ is a parameter which prioritizes the influence of the recent performance.

$$f_1(h_j) = \sum_n \alpha^{n-1} \frac{I_n(h_j)}{T_n(h_j)} \tag{3}$$

The second measurement, f_2, reflects the dependencies between a consecutive pair of LLHs (see Eq. (4)) where $I_n(h_k,h_j)$ denotes the fitness difference between the newly proposed solution and the current solution, $T_n(h_k,h_j)$ denotes the amount of time taken to propose the new solution when h_j is executed right after h_k, $\beta \in (0,1)$ is a parameter which prioritizes the recent performance. Both f_1 and f_2 are the intensification component of the choice function. They encourage the selection of high performance LLHs.

$$f_2(h_k, h_j) = \sum_n \beta^{n-1} \frac{I_n(h_k,h_j)}{T_n(h_k,h_j)} \tag{4}$$

The third measurement, f_3, records the elapsed time since the last execution of a particular LLH (see Eq. (5)) where $\tau(h_j)$ denotes the elapsed time (in seconds) since the last execution of h_j. Note that f_3 acts as a diversification component in the choice function. It prioritizes those LLHs that have not been used for a long time.

$$f_3(h_j) = \tau(h_j) \tag{5}$$

The score of each LLH is computed as a weighted sum of the three measurements, f_1, f_2, and f_3, as shown in Eq. (6) where α, β and δ are the respective weight of f_1, f_2, and f_3. In the initial model [13], these parameters were statically fixed. Promising results have been reported when the proposed choice function (i.e. Eq. (6)) is paired with AM as its move acceptance method.

$$F(h_j) = \alpha f_1(h_j) + \beta f_2(h_k, h_j) + \delta f_3(h_j) \tag{6}$$

The parameters in [13] need to be tuned and pre-determined. In order to have a more effective version of the hyper-heuristic, the parameters can be dynamically controlled during execution, as shown in [37]. The values of α and β increases when the selected LLH is able to improve the solution. The growth is proportional to the magnitude of improvement over the previous solution. On the other hand, if the selected LLH performs a non-improving move, α and β are decreased proportionally to the fitness difference. This strategy is shown to be improving the model proposed in [13].

However, Drake, et al. [14] stated some limitations of the strategy in [37]. Firstly, rewarding/penalizing the LLH proportionally to the fitness difference over the previous solution is arguable. During the early stages of optimization, it is easier for a relatively weaker heuristic to obtain a great improvement from a poor starting solution, and a greater reward is assigned to this weaker heuristic. On the other hand, the improvement made in the later stages of optimization is minor (due to convergence to an optimum solution, either local or global), and a lower reward is assigned. However, the improvement made in the later stages is more significant than the improvement made in the early stages, therefore this rewarding scheme might be misleading. Besides that, if no solution can achieve improvement for a number of iterations, this LLH selection method can descend into random selection due to the low α and β settings (i.e. the diversification component, f_3, dominates the score).

Targeted at these limitations, Drake, et al. [14] proposed a variants of choice function, namely MCF, to manage its parameters. In MCF, α and β are combined as a single parameter, μ. The score, F, is computed as follows (Eq. (7)):

$$F_t(h_j) = \mu_t[f_1(h_j) + f_2(h_k, h_j)] + \delta_t f_3(h_j) \tag{7}$$

If the selected LLH yields an improvement, intensification is prioritized by setting μ to a static maximum value close to one, at the same time δ is reduced to a static minimum value close to zero. When the selected LLH fails to improve the solution, μ is penalized linearly with a lower bound of 0.01, while δ grows at the same rate. This prevents the intensification components (i.e. f_1 and f_2) from losing their influence too quickly. Specifically, μ and δ are computed as follows:

$$\mu_t(h_j) = \begin{cases} 0.99, & d > 0 \\ \max[0.01, \mu_{t-1}(h_j) - 0.01], & d \le 0 \end{cases} \tag{8}$$

$$\delta_t(h_j) = 1 - \mu_t(h_j) \tag{9}$$

where d denotes the fitness difference between the newly proposed solution and the previous solution. μ and δ is updated after every LLH execution.

In the subsequent section, the proposed model which integrates the MCF in the ABC algorithm is presented. The main function of MCF is to help the employed and onlooker bees to select an appropriate neighbourhood search heuristic (i.e. LLH).

III. The Proposed Method

The pseudo code of the proposed MCF-ABC model is shown in Algorithm I. MCF-ABC is divided into three phases, i.e. the employed bee phase, onlooker bee phase, and scout bee phase. The proposed MCF-ABC model is a bee algorithm with multiple neighbourhood search heuristics (i.e. LLHs). Five mutational LLHs available in HyFlex for TSP are integrated, namely Random Insertion, Random Swap, Shuffle, Shuffle Subsequence, and Random 2-opt Move. In order for an employed bee or an onlooker bee to select an appropriate LLH, it is aided by MCF as explained in Section II. The details of the five LLHs are as follows:

- Random Insertion (RI): Randomly pick a city from a solution, remove it from the solution, and reinsert it to a random position of the solution.
- Random Swap (RS): Swap the position of two randomly selected cities in a solution.
- Shuffle (S): Re-order all the cities at random.
- Shuffle Subsequence (SS): Re-order a randomly selected subsequence of cities at random.
- Random 2-opt Move (R2opt): Break two randomly selected arcs and reconnect them in another possible way.

Each LLH (i.e. RI, RS, S, SS, and R2opt) has a score, F. Each employed bee or onlooker bee selects a LLH based on the F score (lines 10 and 22 in Algorithm I). The computation of F is shown in Eq. (7). The LLH with the largest F score is selected. After performing a neighbourhood search, the F score of the selected LLH is updated using Eq. (7) (lines 18 and 30 in Algorithm I).

As MCF-ABC is implemented using HyFlex and HyFlex uses the execution time as the main factor in comparison studies, the stopping criterion of MCF-ABC is adapted such that it is based on the execution time as well. In other words, the algorithm is executed for a maximum amount of time, in this case, denoted by t_{max}, as shown in line 8 in Algorithm I. Note that t_{max} is determined by a benchmarking software obtained from the CHeSC website [25]. The benchmarking software is able to test the speed of the machine which performs the operations involved in iterative hyper-heuristics for combinatorial optimization. It determines the maximum length of the execution time of a particular machine to run a particular algorithm.

As for the solution (food source) abandonment criterion, MCF-ABC applies the same mechanism as proposed in the original ABC algorithm by Karaboga [3]. If a particular solution (i.e. food source) has not been improved after the *limit* trials, it is abandoned. The employed bee associated to the abandoned food source becomes a scout bee, and it goes to source for a new food source (i.e. solution) at random. The scout bee uses the initialization procedure provided by HyFlex to generate a new solution (line 34 in Algorithm I).

Algorithm I: Pseudo code of the MCF-ABC model.

```
1 Procedure MCF-ABC()
  //initialization
2   Initialize maxIteration & popsize
3   limit = popSize / 2 x dim
4   for i = 1 to popSize/2
5     foodSource[i] = initializeSolutions()
6     foodSource[i].counter = 0
7   end for
8   while not reaching maxIteration do
      //Employed bee phase
9     for i = 1 to popSize/2
10      selectedLLH = selectLLH_BasedOnMCF()
11      newSol = neighbourSeach(foodSource[i], selectedLLH)
12      if getFitness(newSol) < getFitness(foodSource[i])
13        foodSource[i] = newSol
14        foodSource[i].counter = 0
15      else
16        foodSource[i].counter++
17      end if
18      updateChoiceFunction(selectedLLH) //eq. (7)
19    end for
      //Onlooker bee phase
20    for i = 1 to popSize/2
21      k = selectSolBasedOnRW(foodSource)
22      selectedLLH = selectLLH_BasedOnMCF()
23      newSol = neighbourSeach(foodSource[k], selectedLLH)
24      if getFitness(newSol) < getFitness(foodSource[k])
25        foodSource[k] = newSol
26        foodSource[i].counter = 0
27      else
28        foodSource[i].counter++
29      end if
30      updateChoiceFunction(selectedLLH) //eq. (7)
31    end for
      //Scout bee phase
32    for i = 1 to popSize/2
33      if foodSource[i].counter > foodSource[i].limit
34        foodSource[i] = initializeSolutions()
35        foodSource[i].counter = 0
36      end if
37    end for
38  end while
39 end Procedure
```

IV. Results and Discussion

The experimental setting, experimental results, and comparison studies are presented in this section.

A. Experimental Setting and Results

All experiments were 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 was executed by one processor only.

The algorithm stopping criteria were set as follows. Two sets of experiments with different stopping criteria were designed to analyze the performance of MCF-ABC. The first stopping criterion was based on the execution time. Based on the benchmarking software obtained from the CHeSC website [25], t_{max} was set at 346 seconds for the machine used in our experimentation. The second stopping criterion was based on the execution iterations (i.e. 100000 iterations).

For both sets of experiments, two parameters of MCF-ABC were pre-determined, i.e. the population size, *popSize*, and the number of trials for improvement of a particular solution before it was abandoned, *limit*. For comparison purpose, the parameters were set based on [3]: *popSize*=20 and *limit*=*popSize*/2**dim*, where *dim* denotes the TSP dimension (i.e. number of cities). Since the *popSize* was set at 20, there would be *popSize*/2=10 employed bees and 10 onlooker bees.

MCF-ABC was executed for five replications for each of the ten TSP instance available in HyFlex. The best, average, the worst tour lengths and their corresponding deviation percentages, *d*, for each instance were recorded. Table I shows the results obtained based on the time-based stopping criterion (i.e. 346 seconds), while Table II shows those based on the iteration-based stopping criterion (i.e. 100000 iterations).

Table I: The MCF-ABC performance on ten TSP instances in HyFlex after an execution time of 346 seconds.

Instances	Best		Average		Worst	
	Tour Length	*d* (%)	Tour Length	*d* (%)	Tour Length	*d* (%)
pr299	50632.90	5.07	51076.77	5.99	52006.18	6.78
pr439	113226.73	5.61	114101.21	6.42	118307.06	7.45
rat575	7167.15	5.82	7189.90	6.16	7430.36	6.48
u724	44618.10	6.46	44843.00	7.00	47327.79	7.78
rat783	9391.47	6.65	9440.72	7.21	10064.66	7.48
pcb1173	62149.20	9.24	62443.74	9.76	68371.71	10.35
d1291	54913.38	8.10	55352.25	8.96	58906.62	10.11
u2152	72803.89	13.31	73547.83	14.47	77092.50	15.20
usa13509	24628339.76	23.25	24743505.58	23.82	24840729.29	24.31
d18512	784401.62	21.57	785913.33	21.80	787439.07	22.09

Table II: The MCF-ABC performance on ten TSP instances in HyFlex after execution of 100000 iterations.

Instances	Best		Average		Worst	
	Tour Length	*d* (%)	Tour Length	*d* (%)	Tour Length	*d* (%)
pr299	50989.99	5.81	51508.90	6.88	52006.18	7.92
pr439	115193.44	7.44	116391.73	8.56	118307.06	10.34
rat575	7360.49	8.67	7404.52	9.32	7430.36	9.71
u724	46782.74	11.63	47079.93	12.34	47327.79	12.93
rat783	9870.74	12.09	9952.97	13.02	10064.66	14.29
pcb1173	66927.01	17.64	67678.84	18.96	68371.71	20.18
d1291	57590.25	13.36	58319.62	14.80	58906.62	15.96
u2152	75349.87	17.27	76246.15	18.67	77092.50	19.98
usa13509	24628339.76	23.25	24742992.35	23.82	24840729.29	24.31
d18512	784364.44	21.56	785768.78	21.78	787439.07	22.04

B. Comparison Studies

The proposed MCF-ABC model was compared with five ABC variants with a single neighbourhood search heuristic. These five ABC-variants, namely ABC-RI, ABC-RS, ABC-S, ABC-SS, and ABC-R2opt, were implemented using HyFlex with identical experimental settings, in order to ensure a fair comparison. The details of these neighbourhood search heuristics, i.e. RI, RS, S, SS, and R2opt, can be found in Section III.

To compare the performance among MCF-ABC and five ABC variants, the Friedman test was conducted, with the following null hypothesis (H0) and alternative hypothesis (HA): H0: all these models have identical performance; HA: at least one model has significantly different performance from another model. The confidence interval was set at 95%. Regardless of the stopping criterion (i.e. whether time-based or iteration-based), when p-value $\approx 0.000 << 0.05$, the null hypothesis would be rejected.

The mean ranking (the lower the better) obtained from the Friedman test for each method considered in the comparison study is shown in Table III. MCF-ABC and ABC-R2opt ranked the first, followed by ABC-RI, ABC-RS, ABC-S, and ABC-SS.

The Wilcoxon signed ranks test was also performed to illustrate the pairwise comparisons between MCF-ABC and five ABC variants. The Wilcoxon signed ranks test results are presented in Table IV.

Based on the 95% confidence interval, Table IV indicates that MCF-ABC significantly outperforms ABC-RI, ABC-RS, ABC-S and ABC-SS with *p*-value < 0.05 and $R^+ > R^-$, and it is comparable with ABC-R2opt (i.e. *p*-value > 0.05), regardless of the stopping criterion.

Table III: The mean rankings of the Friedman test.

ABC Variants	Stopping Criterion	
	Time-based	Iteration-based
MCF-ABC	1.60	1.70
ABC-RI	3.10	3.00
ABC-RS	3.80	4.35
ABC-S	5.10	5.10
ABC-SS	5.80	5.15
ABC-R2opt	1.60	1.70

Table IV: The Wilcoxon signed rank test results for the pairwise comparison.

Comparison (MCF-ABC vs …)	Stopping Criterion					
	Time-based			Iteration-based		
	p-value	R^+	R^-	*p*-value	R^+	R^-
ABC-RI	0.009	53	2	0.007	54	1
ABC-RS	0.005	55	0	0.005	55	0
ABC-S	0.005	55	0	0.005	55	0
ABC-SS	0.005	55	0	0.007	54	1
ABC-R2opt	0.96	28	27	0.575	22	33

Two observations can be obtained from the comparison study. Firstly, regardless of the stopping criterion used, the overall performance of MCF-ABC statistically outperforms those ABC variants with RI, RS, S and SS as the neighbourhood search heuristics, and it is comparable with ABC-R2opt. Secondly, R2opt is most likely the best performing neighbourhood search heuristic among the five mutation heuristics provided in HyFlex. The results indicate the effectiveness of MCF for automatically selecting proper neighbourhood search heuristics for the employed and onlooker bees.

V. Summary

One of the crucial components of the ABC model is the neighbourhood search, which is performed by the employed and onlooker bees. When ABC is used to solve combinatorial discrete optimization problems, single or multiple problem-specific perturbative heuristic(s) are adopted as the neighbourhood search mechanism of the employed and onlooker bees. When there are multiple neighbourhood search heuristics, the selection of these heuristics has a dramatic impact on the performance of the ABC optimization model. This article proposes the use of a hyper-heuristic method (namely MCF) to guide the selection of the neighbourhood search heuristics in the ABC model automatically.

The proposed MCF-ABC model has been implemented using the HyFlex platform and evaluated with ten TSP instances. Five ABC variants with single neighbourhood search heuristic have been used for comparison purposes. The outcome indicates that MCF-ABC is able to statistically outperform four out of five ABC variants, and is comparable with the remaining one. These results have been obtained using both the time-based or iteration-based stopping criteria. In summary, it is effective to integrate the proposed hyper-heuristic (i.e. MCF) to aid the employed and onlooker bees in selecting an appropriate neighbourhood search heuristic in the ABC model automatically.

Acknowledgment

The authors gratefully acknowledge the support of the Research University Grant (Grant No: 1001/PKOMP/814274) of Universiti Sains Malaysia for this research.

References

[1] C. Blum and X. Li, "Swarm intelligence in optimization," in *Swarm Intelligence*, ed: Springer, 2008, pp. 43-85.
[2] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, *et al.*, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, pp. 1695-1724, 2013.
[3] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Erciyes University, Engineering Faculty, Computer Engineering Department., Technical report-tr062005.
[4] D. Karaboga and B. Basturk, "On the performance of artificial bee colony (ABC) algorithm," *Applied Soft Computing*, vol. 8, pp. 687-697, 2008.
[5] B. Akay, "Performance analysis of artificial bee colony algorithm on numerical optimization problems," PhD Thesis, Erciyes University, Institute of Science, Department of Computer Engineering, 2009.

[6] P. W. Tsai, J. S. Pan, B. Y. Liao, and S. C. Chu, "Enhanced artificial bee colony optimization," *International Journal of Innovative Computing, Information and Control,* vol. 5, pp. 5081-5092, 2009.
[7] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, "A comprehensive survey: Artificial bee colony (ABC) algorithm and applications," *Artificial Intelligence Review,* vol. 42, pp. 21-57, 2014.
[8] J. Denzinger, M. Fuchs, and M. Fuchs, "High performance ATP systems by combining several AI methods," University of Kaiserslautern, Technical Report, SEKI-Report SR-96-091996.
[9] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of Metaheuristics*, ed: Springer, 2010, pp. 449-468.
[10] E. Özcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intelligent Data Analysis,* vol. 12, pp. 3-23, 2008.
[11] P. Demeester, B. Bilgin, P. De Causmaecker, and G. V. Berghe, "A hyperheuristic approach to examination timetabling problems: Benchmarks and a new problem from practice," *Journal of Scheduling,* vol. 15, pp. 83-103, 2012.
[12] W. G. Jackson, E. Ozcan, and J. H. Drake, "Late acceptance-based selection hyper-heuristics for cross-domain heuristic search," in *Proceedings of the 2013 13th UK Workshop on Computational Intelligence (UKCI)* 2013, pp. 228-235.
[13] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," *Practice and Theory of Automated Timetabling III*, pp. 176-190, 2000.
[14] J. H. Drake, E. Özcan, and E. K. Burke, "An improved choice function heuristic selection for cross domain heuristic search," in *Parallel Problem Solving from Nature-PPSN XII*, ed: Springer, 2012, pp. 307-316.
[15] J. H. Drake, E. Ozcan, and 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.
[16] G. K. Koulinas and K. P. Anagnostopoulos, "A new tabu search-based hyper-heuristic algorithm for solving construction leveling problems with limited resource availabilities," *Automation in Construction,* vol. 31, pp. 169-175, 2013.
[17] K. Z. Zamli, B. Y. Alkazemi, and G. Kendall, "A tabu search hyper-heuristic strategy for t-way test suite generation," *Applied Soft Computing,* vol. 44, pp. 57-74, 2016.
[18] P. Dempster and J. H. Drake, "Two frameworks for cross-domain heuristic and parameter selection using harmony search," in *Harmony Search Algorithm*, ed: Springer, 2016, pp. 83-94.
[19] K. Sim. (2011). *KSATS-HH: A simulated annealing hyper-heuristic with reinforcement learning and tabu-search*. Available: http://www.asap.cs.nott.ac.uk/external/chesc2011/index.html
[20] E. Özcan, M. Mısır, G. Ochoa, and E. K. Burke, "A reinforcement learning: great-deluge hyper-heuristic," *International Journal of Applied Metaheuristic Computing (IJAMC),* vol. 1(1), pp. 39-59, 2012.
[21] D. Falcao, A. Madureira, and I. Pereira, "Q-learning based hyper-heuristic for scheduling system self-parameterization," in *Proceedings of the 2015 10th Iberian Conference on Information Systems and Technologies (CISTI)*, 2015, pp. 1-7.
[22] K. Chakhlevitch and P. Cowling, "Hyperheuristics: Recent developments," *Adaptive and Multilevel Metaheuristics,* pp. 3-29, 2008.
[23] M. Kalender, A. Kheiri, E. Özcan, and E. K. Burke, "A greedy gradient-simulated annealing selection hyper-heuristic," *Soft Computing*, vol. 17, pp. 2279-2292, 2013.
[24] G. Ochoa, M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, *et al.*, "Hyflex: A benchmark framework for cross-domain heuristic search," in *Evolutionary Computation in Combinatorial Optimization*, ed: Springer, 2012, pp. 136-147.
[25] M. Hyde and G. Ochoa. (2011). *The cross-domain heuristic search challenge (CHeSC 2011)*. Available: http://www.asap.cs.nott.ac.uk/chesc2011/.
[26] K. M. Galvani and F. J. Von Zuben, "Hyperlab: A Java framework for the creation and management of hyper-heuristics and problem suites," *XI Encontro Nacional de Inteligência Artificial e Computacional*, 2014.
[27] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research,* vol. 59, pp. 231-247, 1992.
[28] D. L. Applegate, *The traveling salesman problem: a computational study*: Princeton University Press, 2006.
[29] A. Banharnsakun, T. Achalakul, and B. Sirinaovakul, "ABC-GSX: A hybrid method for solving the traveling salesman problem," in *Proceedings of 2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC)*, Fukuoka, 2010, pp. 7-12.
[30] H. Sengoku and I. Yoshihara, "A fast TSP solver using GA on JAVA," in *Proceedings of the Third International Symposium on Artificial Life, and Robotics (AROB III'98)*, 1998, pp. 283-288.
[31] D. Karaboga and B. Gorkemli, "A combinatorial artificial bee colony algorithm for traveling salesman problem," in *Proceedings of the International Symposium on Innovations in Intelligent Systems and Applications (INISTA), 2011*, 2011, pp. 50-53.
[32] M. Albayrak and N. Allahverdi, "Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms," *Expert Systems with Applications,* vol. 38, pp. 1313-1320, 2011.
[33] W. H. Li, W. J. Li, Y. Yang, H. Q. Liao, J. L. Li, and X. P. Zheng, "Artificial bee colony algorithm for traveling salesman problem," in *Proceedings of the Advanced Materials Research*, 2011, pp. 2191-2196.
[34] G. Tao and Z. Michalewicz, "Inver-over operator for the TSP," in *Proceedings of the International Conference on Parallel Problem Solving from Nature*, 1998, pp. 803-812.

[35] L. P. Wong, M. Y. H. Low, and C. S. Choong, "A bee colony optimization algorithm for traveling salesman problem," in *Proceedings of the Second Asia International Conference on Modeling & Simulation*, Kuala Lumpur, 2008, pp. 818-823.
[36] M. S. Kıran, H. İşcan, and M. Gündüz, "The analysis of discrete artificial bee colony algorithm with neighborhood operator on traveling salesman problem," *Neural computing and applications*, vol. 23, pp. 9-21, 2013.
[37] P. Cowling, G. Kendall, and E. Soubeiga, "A parameter-free hyperheuristic for scheduling a sales summit," in *Proceedings of the 4th Metaheuristic International Conference*, 2001, pp. 127-131.