A Hyper-heuristic with a Round Robin Neighbourhood Selection

Ahmed Kheiri and Ender Özcan

University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
{axk,exo}@cs.nott.ac.uk

Abstract. An iterative selection hyper-heuristic passes a solution through a heuristic selection process to decide on a heuristic to apply from a fixed set of low level heuristics and then a move acceptance process to accept or reject the newly created solution at each step. In this study, we introduce Robinhood hyper-heuristic whose heuristic selection component allocates equal share from the overall execution time for each low level heuristic, while the move acceptance component enables partial restarts when the search process stagnates. The proposed hyper-heuristic is implemented as an extension to a public software used for benchmarking of hyper-heuristics, namely HyFlex. The empirical results indicate that Robinhood hyper-heuristic is a simple, yet powerful and general multistage algorithm performing better than most of the previously proposed selection hyper-heuristics across six different Hyflex problem domains.

1 Introduction

A *hyper-heuristic* is a heuristic that performs a search over a space of heuristics, as opposed to space of solutions directly. Although the term hyper-heuristic was coined recently, the idea of combining the strengths of different heuristics (neighbourhood operators) dates back to the 1960s [10]. An aim of the hyper-heuristic research is to raise the level of generality by providing a high level strategy that is applicable across different problem domains rather than a single one. There are two main types of hyper-heuristics in the academic literature: methodologies used for *generation* or *selection* of heuristics [5,6,23].

A selection hyper-heuristic methodology combines heuristic selection and move acceptance processes under a single point search framework [1,2,8,20,21]. A candidate solution is improved iteratively by selecting, applying a heuristic (neighbourhood operator) from a set of low level heuristics, then passing the new solution through move acceptance to replace the solution in hand at each step. [5,7] are the recent surveys on hyper-heuristics. In this study, a new simple selection hyper-heuristic is introduced and tested across a variety of problem domains using Hyflex (Hyper-heuristics Flexible framework) [18], a software tool for hyper-heuristic development and research.

© Springer-Verlag Berlin Heidelberg 2013

1.1 Hyflex

Hyflex provides an interface for the implementation of not only hyper-heuristics but also other (meta)heuristics and problem domains. Any problem domain developed for Hyflex is required to define a set of low level heuristics (neighobourhood operators) which should be classified as *mutational* (MU), *hill climbing* (HC), *ruin and re-create* (RC) or *crossover* (XO). A mutational heuristic makes a random perturbation producing a new solution and this process does not necessarily generate an improvement over the input solution. Local search or hill climbing is often an iterative procedure searching different neighbourhoods starting from a given solution. A ruin and re-create operator produces a partial solution from a given complete solution and then rebuilds a new complete solution. Crossover is a well known operator in evolutionary computation, which takes two solutions and produces a new solution. In general, crossover yields two new solutions and the best new solution is returned in Hyflex.

HyFlex provides utilities to control the behaviour of some low level heuristics to a limited extent. It is possible to increase or decrease the intensity of some mutational and ruin and re-create operations by adjusting a control parameter from 0.0 to 1.0. Changing the value of the intensity parameter could mean changing the range of new values that a variable can take in relation to its current range of values or changing the number of solution variables that will be processed by a heuristic. There is also another similar control parameter for some local search operators for changing the depth of search which relates to the number of hill climbing steps.

HyFlex currently provides an implementation of six minimisation problem domains: Boolean Satisfiability (MAX-SAT), One Dimensional Bin Packing (BP), Permutation Flow Shop (PFS), Personnel Scheduling (PS), Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP) each with different instances and a set of low-level heuristics. The nature of each low level heuristic for each Hyflex problem domain is summarised in Table 1. Currently, there are 12 different instances for the first four problem domains and 10 for the last two problem domains. What is left is to design and implement a high-level strategy (hyper-heuristic) that intelligently selects and applies suitable low-level heuristics from the set provided to each instant from the given domain to get the minimum objective function value in ten minutes. Hyflex was used for Cross-domain Heuristic Search Challenge (CHESC)[1] in 2011. The hyper-heuristics entered into this competition and made it to the finals serve as a benchmark for newly developed hyper-heuristics targeting generality.

2 Related Work

There is a growing number of work on selection hyper-heuristics which have been designed and tested using hyflex. Before CHeSC 2011 (Cross-Domain Heuristic

[1] `http://www.asap.cs.nott.ac.uk/chesc2011/`

Table 1. The nature of the low level heuristics used in each problem domain. The bold entries for each problem domain mark the last low level heuristic of each type.

Heuristic IDs	LLH0	LLH1	LLH2	LLH3	LLH4	LLH5	LLH6	LLH7	LLH8	LLH9	LLH10	LLH11	LLH12	LLH13	LLH14
MAX-SAT	MU$_0$	MU$_1$	MU$_2$	MU$_3$	MU$_4$	**MU**$_5$	**RC**$_0$	HC$_0$	**HC**$_1$	XO$_0$	**XO**$_1$				
Bin Packing	MU$_0$	RC$_0$	**RC**$_1$	MU$_1$	HC$_0$	**MU**$_2$	**HC**$_1$	**XO**$_0$							
PS	HC$_0$	HC$_1$	HC$_2$	HC$_3$	**HC**$_4$	RC$_0$	RC$_1$	**RC**$_2$	XO$_0$	XO$_1$	**XO**$_2$	**MU**$_0$			
PFS	MU$_0$	MU$_1$	MU$_2$	MU$_3$	**MU**$_4$	RC$_0$	**RC**$_1$	HC$_0$	HC$_1$	HC$_2$	**HC**$_3$	XO$_0$	XO$_1$	XO$_2$	**XO**$_3$
TSP	MU$_0$	MU$_1$	MU$_2$	MU$_3$	**MU**$_4$	**RC**$_0$	HC$_0$	HC$_1$	**HC**$_2$	XO$_0$	XO$_1$	XO$_2$	**XO**$_3$		
VRP	MU$_0$	MU$_1$	RC$_0$	**RC**$_1$	HC$_0$	XO$_0$	**XO**$_1$	**MU**$_2$	HC$_1$	**HC**$_2$					

Search Challenge), a mock competition was organised with hyflex and the performance of several well known previously proposed hyper-heuristics were compared across a subset of CHeSC problem domains. Burke et al. [3] reported that the best performing hyper-heuristic was an iterated local search approach which applied a randomly selected mutational and ruin and re-create heuristic and then the hill climbers in a predefined sequence. This framework is based on the most successful hyper-heuristic framework reported to perform the best in [21]. Özcan and Kheiri [22] provide a greedy heuristic selection strategy named *dominance-based heuristic selection* which aims to determine low level heuristics with good performance based on the trade-off between the change (improvement) in the solution quality and the number of steps taken. The approach attempts to reduce the number of low level heuristics. It performs well with respect to the mock competition hyper-heuristics on four problem domains of HyFlex. Nguyen et al. [17] tested an evolutionary approach to generate hyper-heuristics across three HyFlex problem domains and according to the experimental results, they obtained only one improving solution over the top two hyper-heuristics from the mock competition.

In the mock competition, each algorithm was run once for each instance, while 31 runs were performed in CHeSC and the median results were compared to determine the best performing hyper-heuristic among the CHeSC participants generalising well across all six (four public and two hidden) HyFlex problem domains given 10 minutes of execution time per instance. The algorithm description of the approaches developed by CHeSC competitors are provided in (http://www.asap.cs.nott.ac.uk/external/chesc2011/results.html). The winner of the competition, denoted as *AdapHH* was a hyper-heuristic which combines a learning adaptive heuristic selection method with an adaptive iteration limited list-based threshold move accepting method ([15]). This hyper-heuristic does not make use of the type information provided for the low level heuristics.

A hyper-heuristic based on Variable Neighborhood Search (*VNS-TW*) ranked the second at the competition [11]. This approach applies shaking heuristics followed by hill-climbers to solve the problems. The third ranking algorithm (*ML*) was based on a self-adaptive meta-heuristic using multiple agents. The fourth approach (*PHUNTER*) was inspired by the pearl hunting utilising two phases *diversification* and *intensification*. The fifth hyper-heuristic (*EPH*) was based on evolutionary programming which evolves population of solutions and heuristics. The other hyper-heuristics were inspired from well known population based and single point-based search metaheuristics: Iterated Local Search driven by Evolutionary Algorithms (*ISEA*) [14], Hybrid Adaptive Evolutionary Algorithm (*HAEA*), Genetic Hive Hyper-heuristic (*GenHive*), Ant Colony Optimization based hyper-heuristic (*ACO*), Simulated Annealing Hyper-Heuristic with Reinforcement Learning and Tabu-Search (*KSATS-HH*), Reinforcement Learning (*AVEG-Nep*) [9] and Generic Iterative Simulated-Annealing Search (*GISS*). More on these approaches can be found at the competition webpage.

After the CHeSC 2011 competition, a number of researchers attempted to improve previously proposed hyper-heuristics. Kalender et al. [13] proposed a hyper-heuristic which combines a learning greedy gradient approach for heuristic selection and simulated annealing move acceptance. The results show that this approach performs slightly better than a Choice Function hyper-heuristic whose performance is improved by Drake et al. [12] substantially with a minor adjustment. Although, these approaches improved the performance of the traditional Choice Function and Greedy hyper-heuristics on HyFlex problem domains, their performances still on average as compared to the hyper-heuristics of CHeSC competitors. In [4,19], the authors proposed an adaptive neighbourhood iterated local search algorithm based on Hyflex and its variant.

The proposed hyper-heuristic in this study is also implemented as an extension to HyFlex. Its performance is compared to the mock competition hyper-heuristics as well as hyper-heuristics provided by the CHeSC competitors.

3 Methodology

This study introduces a multi-stage selection hyper-heuristic framework based on a **ro**und-ro**bin** neighbour**hood** selection mechanism (Algorithm 1). This framework gives a chance for each low level heuristic in a selected subset of low level heuristics to execute for a certain duration at a stage. A low level heuristic is chosen in a round robin fashion. Depending on the strategy whole set of low level heuristics can be used and the order of low level heuristics can be fixed or varied. Any move acceptance method could be used within this framework. We describe an easy-to-implement yet powerful selection hyper-heuristic based on this framework which will be referred to as *Robinhood hyper-heuristic* in this section. The Robinhood hyper-heuristic is implemented for Hyflex accommodating performance testing across domain implementations and comparison to the top hyper-heuristics competed in CHeSC.

The Robinhood hyper-heuristic is composed of components inspired from previously proposed approaches. The heuristic selection methods presented by

Algorithm 1. Robinhood hyper-heuristic framework

```
1: procedure Robinhood
2:     initialise();
3:     while (terminationCriteriaNotSatisfied1()) do    ▷ e.g., terminate when the given overall execution time is exceeded
4:         update1(); ▷ set/update relevant parameter/variable values before entering into a stage or no-op
5:         for ($i$ =nextLowLevelHeuristicID()) do    ▷ entry of the stage
6:             while ( terminationCriteriaNotSatisfied2() ) do ▷ e.g., terminate when the given time for a heuristic is exceeded
7:                 $S'$ =applyLLH($i, S$);  ▷ $S$ and $S'$ are the current and new solutions, respectively
8:                 moveAcceptance($S, S'$);
9:             end while
10:            update2();    ▷ set/update relevant parameter/variable values after employing a low level heuristic or no-op
11:        end for
12:        update3();   ▷ set/update relevant parameter/variable values after a stage or no-op
13:    end while
14: end procedure
```

Cowling et al. [8] includes *Random Permutation* and *Random Permutation Gradient*. This method applies a low level heuristic one at a time sequentially in a randomly generated permutation order. Random Permutation Gradient operates in the same with a minor change that is as long as the chosen heuristic makes an improvement in the current solution the same heuristic is employed. Given a time limit of t (Algorithm 1, line 3), and n low level heuristics, the Robinhood hyper-heuristic fixes the number of stages as k and applies all low level heuristics (line 5) to the current solution in a given order for $t/(n.k)$ time unit at a stage (line 6). Hyflex does not provide any annotation for the low level heuristics in a given domain, indicating whether they operate on a given solution with a stochastic or deterministic approach. Although this could be detected with a level of certainty using some initial tests over the set of heuristics, we assumed that all operators are stochastic and so each heuristic is given an equal amount of time to process a given solution at each stage.

We classified all ruin and re-create low level heuristics provided in a given problem domain as mutational heuristics, since Hyflex does not provide any indication whether a ruin and re-create heuristic is a mutational or local search operator. The proposed hyper-heuristic aims to use all low level heuristics assuming that the domain implementers chose reasonable heuristics which will not be misleading for the search process. Consequently, we randomly order the low level heuristics within each group of heuristics: mutational, crossover and hill climbing. Inspired by memetic algorithms [16,24], in which solutions are improved through successive application of mutation, crossover and hill climbing, the Robinhood hyper-heuristic uses the same ordering of groups and randomly fixing the ordering of heuristics within each group at a stage. There is also a strong empirical evidence in the literature that this ordering is a good choice even for selection hyper-heuristics as reported in [3,21]. Our hyper-heuristic uses the same ordering in the subsequent stage if there is an improvement in the solution quality at a given stage. Otherwise, without changing the group ordering, another random ordering of low level heuristics within each group is generated for the subsequent stage.

In this study, we discretised the choices for the control parameters provided in Hyflex into five different levels of intensity and depth of search: {0.1, 0.3, 0.5, 0.7, 0.9}. A low level heuristic with the chosen parameter setting is treated as a different heuristic, hence producing five heuristics instead of one. The crossover operator is not commonly used by single point-based search techniques. In order to be able to apply a crossover heuristic, an extra solution (argument) is required. In our approach, the current solution is always used as one of the solutions passed to the crossover operator. To decide on the second solution, we used a circular queue containing M best solutions so far. Again, the round robin strategy is employed. A pointer is used to indicate which solution will be used from this queue during crossover. After a crossover operation, the pointer advances to the next item in the list for the next crossover.

A modified version of the adaptive acceptance method in [3] is introduced for the move acceptance. This acceptance method accepts all improvements as usual, but the deteriorations are accepted with respect to an adaptively changing rate, denoted as *acceptanceRate*. Assuming a minimisation problem, let $f(x)$ denote the quality of a given solution x, then if $f(S')$ is less than $f(S)$, then S' is accepted, otherwise S' is accepted with a uniform probability of acceptanceRate (Algorithm 1, line 8). Initially, only strictly improving moves are accepted. However, if the solution does not improve for one stage, only the moves generating improving or equal quality new solutions are accepted. If the solution does not improve for another following stage, then threshold move acceptance is activated based on acceptanceRate. A reinforcement learning mechanism is used for adapting the value of acceptanceRate. If the solution gets stuck at a local optimum for a given stage, then acceptanceRate is increased by a δ value for the next stage, making it more likely that a worsening solution is accepted. Conversely, if the solution in hand worsens in a given stage, then the acceptanceRate is reduced by δ, making it less likely for a worsening solution to be accepted. the value of δ is fixed arbitrarily as 0.01 during the experiments. The acceptanceRate value updates are intended to help the search navigate out of local optima, and focus the search if it is progressing well.

4 Empirical Results

In all the cases, a run terminates after $t = 600$ seconds or equivalent to 10 minutes as the competition requires. The equivalent value is obtained using the benchmarking tool provided at the competition website. Initial experiments are performed for parameter tuning of number of stages, k. Testing different values of $k = \{1, 2, 10, 20, 30, ..., 100, 200, 300, 1000\}$ revealed that the best k value is 200 in the current. Due to the memory limitation, the size of the stored solutions M has fixed arbitrarily as 50. The Formula1 scoring system is used for comparing the performance of hyper-heuristics. The top hyper-heuristic receives 10 points, the second one gets 8 and then 6, 5, 4, 3, 2, 1, respectively. The remaining approaches get zero point. These points are accumulated as a score for a hyper-heuristic over all instances.

4.1 Performance Comparison to the Mock Competition Hyper-heuristics

The performance of the Robinhood hyper-heuristic (RHH) is compared to the performances of eight different previous hyper-heuristics (HH1-HH8) across four problem domains, each with 10 different instances, as provided for the mock competition at: www.asap.cs.nott.ac.uk/external/chesc2011/defaulthh.html The problem domains used in the mock competition are Boolean Satisfiability (MAX-SAT), One Dimensional Bin Packing (BP), Personnel Scheduling (PS) and Permutation Flow Shop (PFS). A single run is performed using each problem instance in the mock competition.

The Robinhood selection hyper-heuristic outperforms the mock competition hyper-heuristics with a Formula 1 score of 264.25 in the overall (Figure 1). It obtains the best results in 7 out of 10 instances with 2 draws in the MAX-SAT and with no draws in the 1D Bin Packing. In the personnel scheduling problem, RHH produces the best results in 2 instances. RHH delivers a relatively poor performance in the permutation flow shop problem domain as compared to its performance on the other domains. RHH is the winner in the MAX-SAT and 1D Bin Packing problem domains and looses to the other hyper-heuristics in the rest of the problem domains.

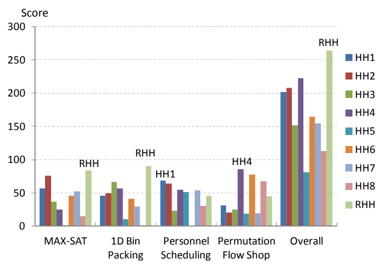

Fig. 1. Comparisons of the different hyper-heuristics over each domain based on Formula1 scores

4.2 Analysis of RHH and Its Performance Comparison to the CHESC Competitors

The Robinhood selection hyper-heuristic is run for 31 times across six problem domains including Boolean Satisfiability (MAX-SAT), One Dimensional Bin Packing (BP), Personnel Scheduling (PS), Permutation Flow Shop (PFS), Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP). We have initially analysed whether the proposed heuristic selection method makes effective use of the low level heuristics. Figure 2 shows the *percentage utilisation* of the low level heuristics considering improving moves only with respect to the number of times a heuristic gets selected while solving an arbitrary instance from each problem domain as an example. A similar phenomena is observed for the other instances. Not all the low level heuristics are useful in improving a solution. For example, in 1D Bin Packing, LLH1, LLH5 and LLH7 generates no improvements. Most of the improving moves are due to mutational heuristics rather than hill climbers across all problem domains. Ruin and recreate heuristics are more successful for creating improving moves in the Permutation Flow Shop domain, while a hill climbing heuristic creates the most improvements in 1D bin packing problem domain. Although a heuristic that does not generate any improvement could still be useful when used in combination with another heuristic, so there is a lot of research scope for methodologies attempting to reduce the number of low level heuristics before and during a run.

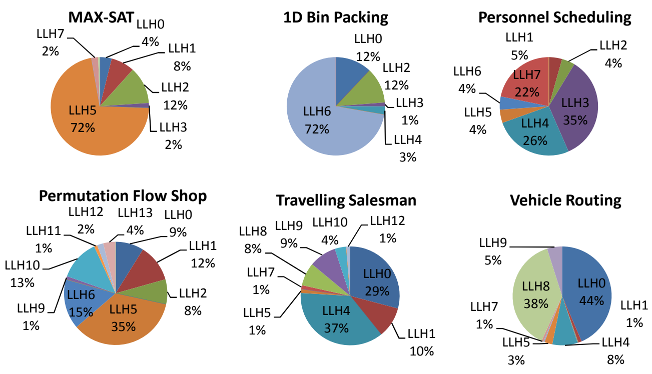

Fig. 2. Percentage utilisation of low level heuristics obtained from a sample run while solving an arbitrary instance from each problem domain

We have investigated the behavior of RHH based on the proposed acceptance method. In most of the cases, RHH rapidly improves the quality of the solution in hand. After a while, the improvement process slows down as the approach reaches a local optimum. Still, it seems that the proposed move acceptance works well as a part of the proposed hyper-heuristic, allowing further improvement in time even if takes a while to obtain an improved solution. The proposed move acceptance allows partial restarts and the extension of these restarts change if there is no improvement and in general there is some improvement. This behaviour is illustrated in Figure 3 for an arbitrarily selected instance from each problem domain. Similar phenomena are observed in the other instances. RHH seems to require partial restarts while solving problems from the MAX-SAT, Permutation Flow Shop and Personnel Scheduling problem domains more than the others. For the Vehicle Routing and somewhat 1D Bin Packing problem domains, RHH generates continuous improvement via the heuristics.

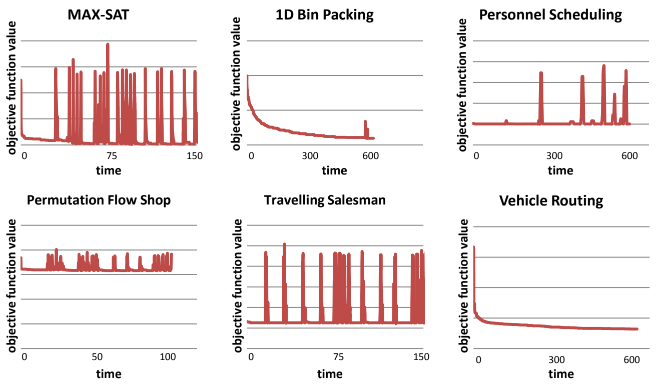

Fig. 3. Plots of the objective value versus time from a sample run while solving an arbitrary instance from each problem domain

The performance of the Robinhood selection hyper-heuristic is compared to the performances of CHeSC hyper-heuristics as provided at the competition website. The comparison is based on the Formula1 scoring system using the median of 31 runs for each instance. The points are accumulated as a score for each hyper-heuristic over five instances from six problem domains including Boolean Satisfiability (MAX-SAT), One Dimensional Bin Packing (BP), Personnel Scheduling (PS), Permutation Flow Shop (PFS), Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP). Table 2 summarises the results. The table shows that RHH is ranking the fourth when compared to the algorithms implemented by the CHeSC competitors with a total score of 93.70.

The normalized function values based on the median of 31 runs for each instance can also be used to evaluate the performance of the different hyper-heuristics [9]. The objective values are calculated and rescaled in [0,1] as a score to rank the different approaches for each problem domain. Figure 4 illustrates the normalized function for the problem domains in which Robinhood hyper-heuristic performs the best and worst. The results are still consistent with the previous findings. The Robinhood hyper-heuristic is the top in the VRP problem domain. In the MAX-SAT and 1D Bin Packing problem domains, the RHH produces good quality of solutions comparing to other approaches. In the other domains, RHH produces a relatively poor performance. It delivers the worst performance, particularly in the permutation flow shop problem domain.

Table 2. Formula 1 scores of the top ten hyper-heuristics among CHeSC finalists and Robinhood hyper-heuristic (RHH) across six problem domains

HH	SAT	BP	PS	PFS	TSP	VRP	TOT
AdapHH	33.10	45.00	8.00	37.00	40.25	11.00	174.35
VNS-TW	33.60	2.00	37.50	34.00	16.25	4.00	127.35
ML,	11.00	8.00	29.50	39.00	13.00	21.00	121.50
RHH	**22.70**	**26.00**	**16.00**	**0.00**	**3.00**	**26.00**	**93.70**
PHUNTER	8.00	3.00	11.50	9.00	26.25	29.00	86.75
EPH	0.00	7.00	9.50	21.00	36.25	12.00	85.75
HAHA	31.10	0.00	23.50	3.50	0.00	13.00	71.10
NAHH	11.50	19.00	1.00	22.00	12.00	5.00	70.50
ISEA	3.50	28.00	14.50	3.50	11.00	4.00	64.50
KSATS-HH	21.70	7.00	7.50	0.00	0.00	19.00	55.20

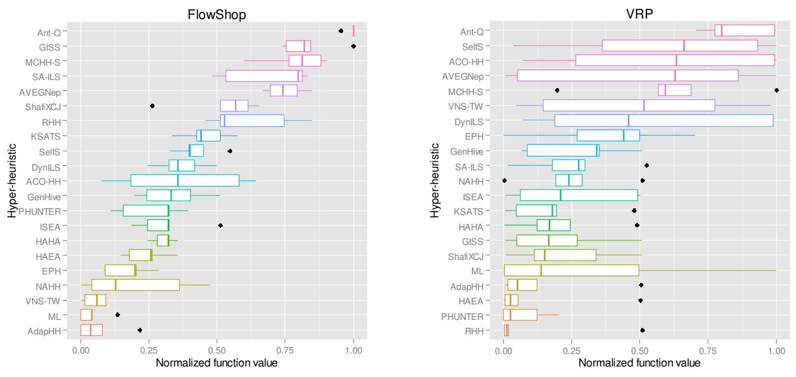

Fig. 4. Normalized function values

5 Conclusion and Future Work

Hyper-heuristics have been shown to be effective solution methods across many problem domains. In this study, an easy-to-implement selection hyper-heuristic combining a round-robin strategy-based neighbourhood selection and an adaptive move acceptance methods is introduced. The Robinhood hyper-heuristic allocates equal share from the overall time for each low level heuristic ordering them randomly within their categories of mutation, crossover and local search. In this manner, memetic algorithm is imitated under a single point-based search framework with multiple operators. The Robinhood hyper-heuristic operates in stages and prior to each stage, relevant decisions are made for the ordering of heuristics within groups and parameters of the system components, such as move acceptance. The experimental results show that proposed hyper-heuristic is a simple yet very powerful and general strategy outperforming many previously proposed selection hyper-heuristics across six different domains. As for the future work, we plan to work on more learning components within this framework to further improve its performance without introducing additional parameters and making the hyper-heuristic more complicated. The Robinhood hyper-heuristic has only three parameters: δ, M and k. All these values are currently tuned after some experimentation, but of course, the question is whether it is possible to adapt them during the search process, in particular the duration allocated for each stage and get improved performance. We have observed that some of the heuristics are almost useless at different stages of the search process, then by reducing the number of heuristics involved in the search process at a stage would be a good idea.

References

1. Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics, pp. 457–474. Kluwer (2003)
2. Burke, E., Kendall, G., Misir, M., Özcan, E.: Monte carlo hyper-heuristics for examination timetabling. Annals of Operations Research, 1–18 (2010)
3. Burke, E.K., Curtois, T., Hyde, M.R., Kendall, G., Ochoa, G., Petrovic, S., Rodríguez, J.A.V., Gendreau, M.: Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In: IEEE Congress on Evolutionary Computation, pp. 1–8 (2010)
4. Burke, E.K., Gendreau, M., Ochoa, G., Walker, J.D.: Adaptive iterated local search for cross-domain optimisation. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 1987–1994. ACM, New York (2011)
5. Burke, E.K., Genreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. Technical report (to appear)
6. Burke, E., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.: A classification of hyper-heuristics approaches. In: Handbook of Metaheuristics. Springer (2009) (in press)
7. Chakhlevitch, K., Cowling, P.: Hyperheuristics: Recent Developments. In: Cotta, C., Sevaux, M., Sörensen, K. (eds.) Adaptive and Multilevel Metaheuristics. SCI, vol. 136, pp. 3–29. Springer, Heidelberg (2008)
8. Cowling, P., Kendall, G., Soubeiga, E.: A Hyperheuristic Approach to Scheduling a Sales Summit. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 176–190. Springer, Heidelberg (2001)
9. Di Gaspero, L., Urli, T.: Evaluation of a Family of Reinforcement Learning Cross-Domain Optimization Heuristics. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, vol. 7219, pp. 384–389. Springer, Heidelberg (2012)
10. Fisher, H., Thompson, G.L.: Probabilistic learning combinations of local job-shop scheduling rules. In: Muth, J.F., Thompson, G.L. (eds.) Industrial Scheduling, pp. 225–251. Prentice-Hall, Inc., New Jersey (1963)
11. Hsiao, P.C., Chiang, T.C., Fu, L.C.: A vns-based hyper-heuristic with adaptive computational budget of local search. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (June 2012)

12. Drake, J.H., Özcan, E., Burke, E.K.: An Improved Choice Function Heuristic Selection for Cross Domain Heuristic Search. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part II. LNCS, vol. 7492, pp. 307–316. Springer, Heidelberg (2012)
13. Kalender, M., Kheiri, A., Özcan, E., Burke, E.K.: A greedy gradient-simulated annealing hyper-heuristic for a curriculum-based course timetabling problem. In: 2012 12th UK Workshop on Computational Intelligence (UKCI), pp. 1–8 (September 2012)
14. Kubalík, J.: Hyper-Heuristic Based on Iterated Local Search Driven by Evolutionary Algorithm. In: Hao, J.-K., Middendorf, M. (eds.) EvoCOP 2012. LNCS, vol. 7245, pp. 148–159. Springer, Heidelberg (2012)
15. Misir, M., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G.: A new hyper-heuristic implementation in HyFlex: a study on generality. In: Fowler, J., Kendall, G., McCollum, B. (eds.) Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory & Application, pp. 374–393 (August 2011)
16. Moscato, P., Norman, M.G.: A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. In: Proceedings of the International Conference on Parallel Computing and Transputer Applications, pp. 177–186. IOS Press (1992)
17. Nguyen, S., Zhang, M., Johnston, M.: A genetic programming based hyper-heuristic approach for combinatorial optimisation. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 1299–1306. ACM, New York (2011)
18. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., Burke, E.K.: HyFlex: A Benchmark Framework for Cross-Domain Heuristic Search. In: Hao, J.-K., Middendorf, M. (eds.) EvoCOP 2012. LNCS, vol. 7245, pp. 136–147. Springer, Heidelberg (2012)
19. Ochoa, G., Walker, J., Hyde, M., Curtois, T.: Adaptive Evolutionary Algorithms and Extensions to the HyFlex Hyper-heuristic Framework. In: Coello Coello, C.A., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part II. LNCS, vol. 7492, pp. 418–427. Springer, Heidelberg (2012)
20. Özcan, E., Bilgin, B., Korkmaz, E.E.: Hill Climbers and Mutational Heuristics in Hyperheuristics. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 202–211. Springer, Heidelberg (2006)
21. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. Intelligent Data Analysis 12(1), 3–23 (2008)
22. Özcan, E., Kheiri, A.: A hyper-heuristic based on random gradient, greedy and dominance. In: Gelenbe, E., Lent, R., Sakellari, G. (eds.) Computer and Information Sciences II, pp. 557–563. Springer London (2012)
23. Özcan, E., Parkes, A.J.: Policy matrix evolution for generation of heuristics. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 2011–2018 (2011)
24. Özcan, E., Parkes, A.J., Alkan, A.: The interleaved constructive memetic algorithm and its application to timetabling. Comput. Oper. Res. 39(10), 2310–2322 (2012)