Dynamic Heuristic Set Selection for Cross-Domain Selection Hyper-Heuristics

Ahmed Hassan and Nelishia Pillay[0000−0003−3902−5582]

Department of Computer Science, University of Pretoria, South Africa
ahmedhassan@aims.ac.za
nelishia.pillay@up.ac.za

Abstract. Selection hyper-heuristics have proven to be effective in solving various real-world problems. Hyper-heuristics differ from traditional heuristic approaches in that they explore a heuristic space rather than a solution space. These techniques select constructive or perturbative heuristics to construct a solution or improve an existing solution respectively. Previous work has shown that the set of problem-specific heuristics made available to the hyper-heuristic for selection has an impact on the performance of the hyper-heuristic. Hence, there have been initiatives to determine the appropriate set of heuristics that the hyper-heuristic can select from. However, there has not been much research done in this area. Furthermore, previous work has focused on determining a set of heuristics that is used throughout the lifespan of the hyper-heuristic with no change to this set during the application of the hyper-heuristic. This paper investigates dynamic heuristic set selection (DHSS) which applies dominance to select the set of heuristics at different points during the lifespan of a selection hyper-heuristic. The DHSS approach was evaluated on the benchmark set for the CHeSC cross-domain hyper-heuristic challenge. DHSS was found to improve the performance of the best performing hyper-heuristic for this challenge.

Keywords: Dynamic heuristic set selection · Selection perturbative hyper-Heuristics · Cross-domain hyper-heuristics

1 Introduction

Selection hyper-heuristics explore a heuristic space to choose a constructive or perturbative heuristic at each point to create or improve a solution respectively [14]. Previous work has shown that the performance of selection hyper-heuristics is affected by the set of the constructive or perturbative heuristics that the hyper-heuristic selects from [16,11,15]. Generally, the entire set of heuristics available for the problem domain, which we refer to as the *universal set*, is used by the hyper-heuristic, increasing the search space to be explored which may harm the performance of the hyper-heuristic. We hypothesize that the set of heuristics available to the hyper-heuristic to perform selections from, the *active set*, should be different at different points in the lifespan of the hyper-heuristic. We investigate this hypothesis for cross-domain selection perturbative hyper-heuristics.

Previous work in this area has shown the effectiveness of determining an active set at the beginning of the lifespan of the hyper-heuristic, instead of the hyper-heuristic selecting from the entire set of heuristics. In this case, the active set is static, i.e. the same set is used throughout the lifespan of the hyper-heuristic. In [16], fitness landscape measures are used to estimate the performance of heuristics which are subsequently ranked using non-parametric tests. Several active sets are generated by considering the heuristics according to their ranks. In [6], two ranking methods are used to filter out poor and redundant heuristics: the first method is based on the gain (heuristic performance) and correlation, and the second method is based on the z-test. However, there does not appear to be any research into using different active heuristic sets at different points in the lifespan of a hyper-heuristic. We refer to this as *dynamic heuristic set selection* (DHSS).

This study examines DHSS for cross-domain selection perturbative hyper-heuristics. A dominance approach is used to select the active heuristic set at different points in the lifespan of the hyper-heuristic. The approach is evaluated using the CHeSC challenge benchmark set. It is implemented with the best-performing hyper-heuristic for the challenged, namely, FS-ILS [1]. DHSS was found to improve the cross-domain performance of FS-ILS and FS-ILS together with DHSS outperformed state-of-the-art approaches for the challenge. Hence, the contributions of the research presented in this study are:

1. A DHSS approach for selection hyper-heuristics.
2. An investigation of DHSS for cross-domain hyper-heuristics.

The rest of this paper is organized as follows. The following section defines terms used in the paper in the context of the research presented. Section 3 provides an overview of cross-domain hyper-heuristics and the CHeSC challenge. In Section 4, the approach is described in detail. Section 5 presents the experimental setup. Section 6 discusses the performance of DHSS. Section 7 concludes the paper and presents directions for future research.

2 Terminology

This section presents some terminology that will be used in the paper. The *lifespan* of a selection perturbative hyper-heuristic is a single application of the hyper-heuristic comprised of several iterations. The *universal set* refers to the set of all heuristics for a problem domain. The *active set* is the set of heuristics used at a particular point in the lifespan of a hyper-heuristic. This is often a subset of the universal set. A *heuristic* refers to a problem-specific heuristic that is an element of the universal set. A *phase* is a part of the lifespan in which the active set remains fixed and a *phase length* is the number of iterations executed during the phase. A *duration* of a heuristic is the total execution time in milliseconds used by the heuristic.

3 Cross-Domain Hyper-Heuristics

Cross-domain hyper-heuristics aim at achieving a higher level of generality by solving problems across different problem domains [14]. In this case, hyper-heuristics aim to perform well across all the problem domains rather than producing good results for some problems and poor results for others. The CHeSC challenge aimed at promoting research in cross-domain hyper-heuristics [5]. The challenge required a selection perturbative hyper-heuristic to be developed such that it performed well in six problem domains, namely, the boolean satisfiability problem, one-dimensional bin packing problem, personnel scheduling problem, permutation flow shop problem, traveling salesman problem, and vehicle routing problem. The HyFlex framework [13] was developed for the challenge. HyFlex provides an implementation of heuristics, methods for creating initial solutions, and methods for calculating the objective value for each problem domain. For each problem domain, HyFlex provides four types of heuristics. *Mutational heuristics* perturb a given solution at random. *Ruin and recreate heuristics* destroy and rebuild some parts of a given solution. *Local search heuristics* improve a given solution. *Crossover heuristics* recombine parts from two solutions to produce a new solution.

The selection perturbative hyper-heuristics competing in the challenge were ranked and the ranks are added to obtain the overall score. In cases where there is a tie, the corresponding points to the relevant positions are added together and shared equally among all algorithms that tie. The median of the objective value over 31 independent runs is used to rank the competing hyper-heuristics using Formula 1 in which the top 8 methods (hyper-heuristics) receive a score of 10, 8, 6, 5, 4, 3, 2, 1 points respectively and the rest receive no points. The cross-domain score is calculated by adding up all problem-specific scores and the winner is the hyper-heuristic with the highest cross-domain score. A time limit of 600 seconds is used for each run and a benchmark program is provided to estimate the time limit on different machines so that it matches the 600 seconds on the standard machine used in CHeSC.

Research into producing a competitive selection perturbative hyper-heuristic is still ongoing. We provide an overview of the six best performing hyper-heuristics at the time of writing this paper. Fair share iterated local search (FS-ILS) [1] is a hyper-heuristic that outperforms the winner of CHeSC. FS-ILS uses fitness proportionate selection to select heuristics based on their ability to generate accepted solutions and the duration taken to achieve that. FS-ILS employs a randomized local search step and a Metropolis acceptance criterion. A restart strategy is also incorporated to re-initialize the search if it stagnates for very long. In this paper, we show that the cross-domain performance of FS-ILS can be improved if DHSS is used.

The winner of the CHeSC challenge was adapHH [12] which maintains a subset of heuristics for each phase. adapHH selects a heuristic based on a probability calculated by considering the best improvements and the duration of the heuristic. Relay hybridization is used to identify heuristics that work well in pairs. An adaptive threshold acceptance criterion is used to accept worsening solutions.

VNS [8] based on a variable neighborhood search working on a population of solutions was placed second in the challenge. VNS shakes a solution, which is chosen from the population by a tournament selection. A perturbative heuristic is applied to the solution to achieve this. Then, a local search is applied to the solution. A tabu list is used to keep track of worsening perturbative heuristics. The worse solutions in the population are replaced by better solutions generated by the local search.

The (ML) approach developed by Mathieu Larose [13,4] was placed third in the challenge. ML is an iterated local search hyper-heuristics that employ reinforcement learning to select heuristics. The method consists of a diversification step performed by perturbative heuristics, an intensification step performed by local search heuristics, and an acceptance criterion that accepts worsening moves only if the incumbent solution has not improved for several iterations.

Pearl Hunter (PHunter) [3], placed fourth in the challenge, is an iterated local search that mimics pearl hunters. The search involves two steps: diversification (surface moves) and intensification (dive). PHunter can try more than one diversification move if the search is trapped in a "buoy". PHunter performs two types of local searches: "snorkeling", i.e. short sequences of local search and "scuba dive", i.e. intensive local search.

Evolutionary programming hyper-heuristic (EPH) [10], placed fifth in the challenge, co-evolves two populations: a population of sequences of heuristics that are applied to the solutions in the other population. Each sequence consists of one or two perturbative heuristics (mutational and ruin-recreate heuristics) followed by all local search heuristics.

4 Dynamic Heuristic Set Selection (DHSS) Approach

This section describes the DHSS approach. DHSS uses dominance [2] to determine the active set at each point in the lifespan of the hyper-heuristic. The DHSS is described in Algorithm 1.

Algorithm 1 requires the universal set and search-status information from the hyper-heuristic which includes information such as the current iteration, the elapsed time, and the current solution value. The algorithm starts by initializing the active set to include all heuristics as in line 1. In line 2, the set of permanently excluded heuristics is initialized as an empty set. In line 3, the performance history is initialized for all heuristics. The history keeps track of information about each heuristic such as the percentage improvement, the percentage non-improvement, and duration.

The update condition (in line 4 of Algorithm 1) is checked at the start of every iteration of the hyper-heuristic and the active set is updated if it fails to improve the best solution for N_{fail} iterations where N_{fail} is determined as $pf \times wait_{max}$ where $wait_{max}$ is the maximum number of iterations that have elapsed between two consecutive updates of the best solution and pf is a *patience factor* which controls how fast/slow the updates occur. Further, we observed in some instances, $wait_{max}$ can grow very large if the search stagnates for a long

Algorithm 1 DHSS.

```
Require: universal set $\mathcal{U}$, search-status information $\mathcal{I}$ from the hyper-heuristic
 1: $S \leftarrow \mathcal{U}$ // Initialize the active set $S$
 2: $S' \leftarrow \emptyset$ // Initialize the set of permanently removed heuristics
 3: Initialize performance history $\mathcal{P}$ for all heuristics
 4: if canUpdate($\mathcal{I}$) then
 5:     if canRemove($\mathcal{I}$) then
 6:         $S' \leftarrow$ remove($S$, $S'$, $\mathcal{P}$, $\mathcal{I}$)
 7:     end if
 8:     $S \leftarrow$ update($S$, $S'$, $\mathcal{P}$)
 9:     return $S$
10: else
11:     return $S$
12: end if
```

period before the best solution is updated. For this reason, we set a maximum phase length ($phase_{max}$) to ensure that the active set is still updated even if $wait_{max}$ grows very large.

The removal condition (line 5 of Algorithm 1) is $T_{elp}/T_{max} > R_{excl}$ where T_{elp} is the time elapsed since the start of the hyper-heuristic, T_{max} is the total computational time, and R_{excl} is a parameter that controls how soon the permanent heuristic removal is performed. The heuristic removal is executed once during the search at the first update that happens after T_{elp}/T_{max} exceeds R_{excl}.

The removal criterion (line 6 of Algorithm 1) excludes a heuristic permanently only if it has poor *individual performance* and poor *group performance*. The individual performance of a heuristic is measured from its own history such as the ratio between the number of improvements made by the heuristic and the total number of times the heuristic is used. The group performance considers the performance of a heuristic relative to all heuristics such as the ratio between the percentage improvement made by the heuristic and the total percentage improvement made by all heuristics. The individual performance of a heuristic h_i is defined as follows:

$$f_i^{ind} = \underbrace{\alpha_1 \frac{n_i^+}{n_i^+ + n_i^-} - \alpha_2 \frac{n_i^-}{n_i^+ + n_i^-}}_{\text{frequency}} + \underbrace{\alpha_3 \frac{\Delta_i^+}{\Delta_i^+ + \Delta_i^-} - \alpha_4 \frac{\Delta_i^-}{\Delta_i^+ + \Delta_i^-}}_{\text{amount}} \qquad (1)$$

where n_i^+ and n_i^- denote the number of improvements and non-improvements respectively made by heuristic i; Δ_i^+ and Δ_i^- denote the amount of percentage improvement and percentage non-improvement respectively made by heuristic i; and $0 \leq \alpha_i \leq 1, i = 1, 2, 3, 4$ are weights which are used to give some terms more importance. The group performance of a heuristic h_i is defined as follows:

$$f_i^{gp} = \underbrace{\beta_1 \frac{n_i^+}{\sum_{k=1}^{H} n_k^+} - \beta_2 \frac{n_i^-}{\sum_{k=1}^{H} n_k^-}}_{\text{frequency}} + \underbrace{\beta_3 \frac{\Delta_i^+}{\sum_{k=1}^{H} \Delta_k^+} - \beta_4 \frac{\Delta_i^-}{\sum_{k=1}^{H} \Delta_k^-}}_{\text{amount}} \tag{2}$$

where all symbols are as defined in Eq.(1), H is the total number of heuristics, and $0 \leq \beta_i \leq 1, i = 1, 2, 3, 4$ are weights.

We calculate the averages $\bar{f}_{ind}$ and $\bar{f}_{gp}$ for all values of f_i^{ind} and f_i^{gp} respectively. A heuristic h_i is removed permanently only if $f_i^{ind} < \bar{f}_{ind}$ and $f_i^{gp} < \bar{f}_{gp}$. This criterion removes only the worst-performing heuristics that worsen the current solution, do this often, and fail to compensate for this degenerating behavior by producing significant improvements.

The update criterion (line 8 of Algorithm 1) decides which heuristics should be included in the active set. The update criterion utilizes a *measure*, which evaluates the heuristic performance, to determine suitable heuristics for the next phase. In this study, we measure the heuristic performance by the *frequency of improvements* which favors heuristics that generates more improving moves. The active set is updated such that it includes all *dominant* heuristics and excludes all *dominated* heuristics. A heuristic h_i dominates a heuristic h_k if $v_i > v_k$ and $t_i < t_k$ where v_j is the value of heuristic j as determined by the measure and t_j is the duration of heuristic h_j in milliseconds. This criterion does not exclude a worse heuristic unless it has a longer duration than a better heuristic. The reason for considering the duration is that very slow heuristics are not desirable in general.

5 Experimental Setup

This section describes the experimental setup in terms of parameter tuning and technical specifications.

5.1 Parameter Settings

The parameters involved in DHSS are summarized in Table 1. Manual tuning via trial and error is used to determine the best values of these parameters. For each parameter, several values are tried and the best value is chosen as reported in Table 2. For the α_i and β_i $(i = 1, 2, 3, 4)$ in Eq.(1) and Eq.(2), we tried 3 different configurations:

1. Assign more importance to improvements than non-improvements. In this case, we have $\alpha_1 = \alpha_3 = \beta_1 = \beta_3 = 1.0$ and $\alpha_2 = \alpha_4 = \beta_2 = \beta_4 = 0.5$. This configuration achieves the best results.
2. Assign more importance to non-improvements than improvements. In this case, we have $\alpha_1 = \alpha_3 = \beta_1 = \beta_3 = 0.5$ and $\alpha_2 = \alpha_4 = \beta_2 = \beta_4 = 1.0$.

Table 1: Summary of the parameters of DHSS.

Parameter	Brief Description
$phase_{max}$	Phase length.
pf	Patience factor which controls how fast the updates occur.
R_{excl}	Controls when to start removing some heuristics permanently.
$\alpha1, \beta_1$	Weight of frequency of improvement in Eq.(1) and Eq.(2) respectively.
$\alpha2, \beta_2$	Weight of frequency of non-improvement in Eq.(1) and Eq.(2) respectively
$\alpha3, \beta_3$	Weight of percentage improvement in Eq.(1) and Eq.(2) respectively.
$\alpha4, \beta_4$	Weight of percentage non-improvement in Eq.(1) and Eq.(2) respectively.

Table 2: Parameters values

Parameter	Tried Values	Best Value
$phase_{max}$	1, 100, 1000	100
pf	0.2, 0.5, 1.0	0.5
R_{excl}	0.2, 0.5, 1.5	0.2

3. No distinction. In this case, we have $\alpha_i = \beta_i = 1.0$ for $i = 1, 2, 3, 4$.

We observed that no parameter setting leads to best performance in all problem domains. The best values reported in Table 2 generate the best overall cross-domain performance. The most influential parameters are $phase_{max}$ and R_{excl}. A drop in the performance by about 40% is observed between the best and worst value for $phase_{max}$. The worst value of R_{excl} deteriorates the performance by about 24%.

As part of the parameter tuning, we also tried resetting the active set to include all heuristics that are not permanently removed when the search stagnates for a number of iterations exceeding $wait_{max}$. However, this did not lead to improvement; hence it is not included in Algorithm 1.

5.2 Technical Specification

The experiments are executed in Java 8 on the Lengau Cluster of the Center for High-Performance Computing, South Africa. The cluster's operating system is CentOS 7.0. We used two compute nodes to perform 31 independent runs in parallel. Each node has 24 Intel Xeon CPUs (2.6 GHz) and is connected with FDR 56 GHz InfiniBand. The total RAM used is 2GB.

6 Results and Discussion

This section presents and discusses the results of DHSS across the six domains of HyFlex used in the CHeSC Challenge.

Table 3: FS-ILS compared to the top five hyper-heuristics from the CHeSC Challenge.

Method	FS-ILS [1]	adapHH [12]	VNS [8]	ML [13]	PHunter [3]	EPH [10]
Overall score	178.10	161.68	117.18	111.0	81.60	73.60

Table 4: The cross-domain performance of DHSS compared to the top five methods of CHeSC and FS-ILS.

Method	DHSS	FS-ILS [1]	adapHH [12]	VNS [8]	ML [13]	PHunter [3]	EPH [10]
Overall score	178.75	148.85	142.10	98.10	94.75	71.60	59.10

6.1 Performance Verification

In this section, we verify that FS-ILS is the state-of-the-art hyper-heuristic. We respect all CHeSC competition conditions. The results are presented in Table 3. The cross-domain scores are calculated by adding up all individual scores for each problem domain as described in Section 3. Higher scores correspond to better performance. For brevity, the table shows the top five methods. The cross-domain score of FS-ILS confirms that it is indeed the best hyper-heuristic.

6.2 DHSS and CHeSC Contestants

We evaluate the performance of DHSS with respect to the contestants of CHeSC and FS-ILS across the six domains of HyFlex. We respected all the CHeSC rules. The methods are scored using the CHeSC scoring system described in Section 3 where higher scores indicate better performance. Although we compare DHSS to all contestants of CHeSC, for brevity, Table 4 presents the cross-domain scores for the top five methods, and the best hyper-heuristic for CHeSC (FS-ILS). DHSS improves the cross-domain performance of FS-ILS and achieves the highest cross-domain score. This demonstrates the effectiveness of utilizing DHSS to manage the heuristic set for hyper-heuristics.

The per-domain results are presented in Fig. 1 which shows that DHSS achieves the best performance in 3 domains (SAT, PFS, TSP) and the second-best performance in VRP. No hyper-heuristic dominates all other hyper-heuristics in all domains. For each hyper-heuristic, there is at least one domain that poses a challenge to it. For example, DHSS performs poorly in BP, adapHH performs poorly in PS, and EPH is unable to score any points in SAT; hence, its column is not shown in the figure. In all domains, DHSS performs consistently better than FS-ILS except for BP.

6.3 Analysis of DHSS

In the previous section, we empirically demonstrated the effectiveness of DHSS in improving the cross-domain performance of the best hyper-heuristic for the

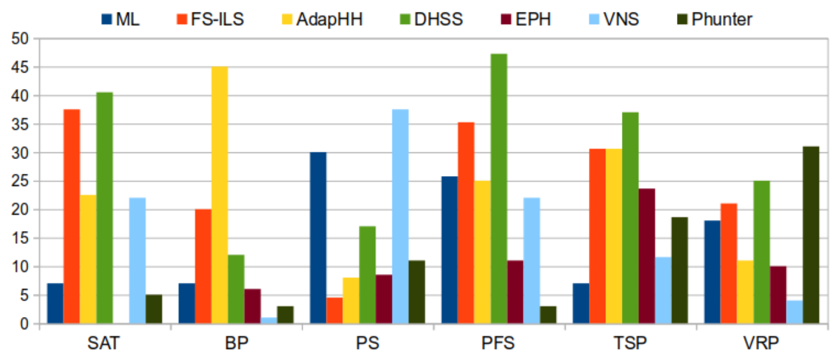


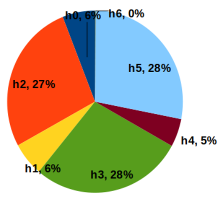
Fig. 1: The performance of DHSS compared to the top five methods of CHeSC and FS-ILS.

CHeSC cross-domain challenge. In this section, we investigate the reasons behind the performance gain. We hypothesize that when DHSS is used, good heuristics will have a larger share of the total computational time, hence utilized more; and poor heuristics will have a smaller share of the total computational time, hence utilized less. To this end, we calculated how much percentage improvement each heuristic contributed to the total percentage improvement. This is measured by the ratio between the percentage improvement made by the heuristic and the total percentage improvement made by all heuristics. We also calculate the share of each heuristic in the total computational time which is measured by the ratio between the total computational time used by the heuristic and the total computational time allocated to the hyper-heuristic.

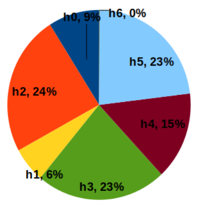
We present the results for two representative domains (SAT and PFS) due to space limitations. From each domain, one instance is chosen arbitrarily. We measured the shares of each heuristic in the total percentage improvement and computational time, as explained above, using 10 runs with different seeds.

The results for SAT are presented in Figures 2 and 3 for the perturbative and local search heuristics respectively. In these figures and all following figures, we use the HyFlex convention in identifying each heuristic by a unique number. These heuristics are described in [9] and [17] for SAT and PFS respectively. The most effective perturbative heuristics for SAT are h2, h3, and h5 which receive collectively an increase of 11% in computational time in DHSS compared to FS-ILS. Furthermore, the local search heuristic h8 is much less effective than h7 since it leads to a marginal improvement. However, in FS-ILS, h8 has an unnecessarily large share of 41% of the total computational time whereas, DHSS restricts the computational time of h8 to 11% of the total computational time.

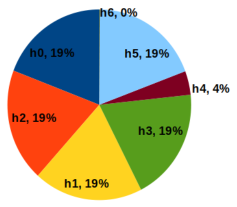
The results of the analysis for PFS are presented in Figures 4 and 5 for perturbative and local search heuristics respectively. The most effective perturbative heuristics are h0, h1, h5, and h6. The heuristic h0 is responsible for more than 50% of the total improvement. The computational time of h0 is 11% larger in



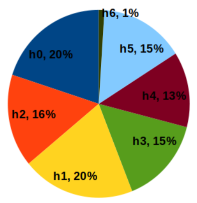
(a) DHSS: Improv.



(b) FS-ILS: Improv.

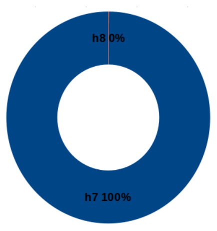


(c) DHSS: Duration.

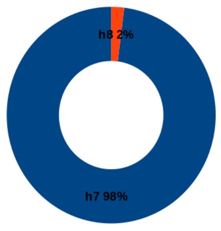


(d) FS-ILS: Durat.

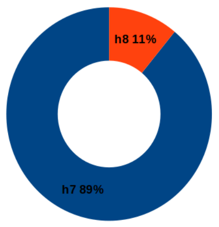
Fig. 2: SAT: Perturbative heuristics.



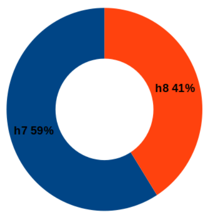
(a) DHSS: Improv.



(b) FS-ILS: Improv.



(c) DHSS: Duration.



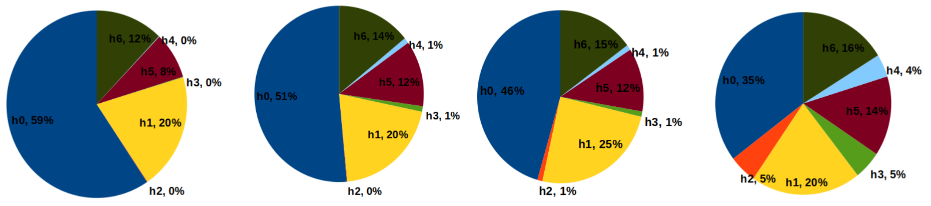
(d) FS-ILS: Duration

Fig. 3: SAT: Local search heuristics.

DHSS compared to FS-ILS. Similarly, h1 has an increase of 5% in computational time in DHSS compared to FS-ILS. On the other hand, h5 and h6 have slightly more computational time in FS-ILS compared to DHSS. In general, good heuristics receive collectively an increase in the computational time of 13% in DHSS compared to FS-ILS. Moreover, poor heuristics (h2, h3, and h4) collectively use 14% of the total computational time in FS-ILS despite collectively contributing by 2% to the total percentage improvement. In DHSS, the collective computational time of h2, h3, and h4 is restricted to only 3% of the total computational time. For local search heuristics, both DHSS and FS-ILS perform well.

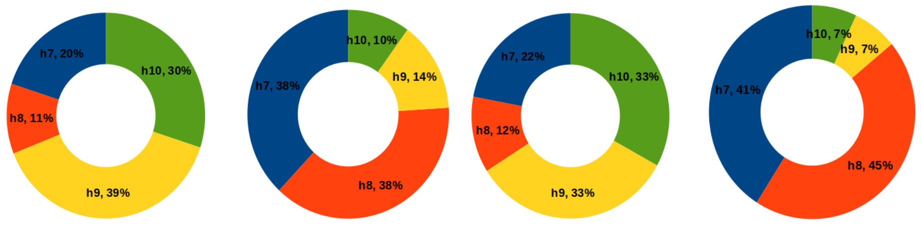
7 Conclusion and Future Work

In this paper, we solved the problem of determining an adequate heuristic set for selection hyper-heuristics dynamically such that the heuristic set changes at different points during the lifespan of the hyper-heuristic. We integrated the proposed approach (DHSS) into the state-of-the-art hyper-heuristic (FS-ILS) and evaluated it across six problem domains. The results indicated that the performance of FS-ILS was improved when the DHSS was used. We carried out an analysis to discover the reasons behind the performance gain. It was found that when the DHSS was used, good heuristics had a larger share of the total computational time, hence utilized more; and poor heuristics had a smaller share of the total computational time, hence utilized less. DHSS can be used



(a) DHSS: Improv. (b) FS-ILS: Improv. (c) DHSS: Duration. (d) FS-ILS: Duration

Fig. 4: PFS: Perturbative heuristics.



(a) DHSS: Improv. (b) FS-ILS: Improv. (c) DHSS: Duration. (d) FS-ILS: Duration

Fig. 5: PFS: Local search heuristics.

with any hyper-heuristics since it does not rely on any specific feature of FS-ILS. We developed an open-source Java library to enable fast development and prototyping of DHSS [7].

In the future, we will consider automating the design of dynamic heuristic set selection approaches. It is also interesting to consider DHSS as a design decision in a wider-scope approach that aims at automating the design of multiple aspects of the hyper-heuristics simultaneously.

Acknowledgment

This work is funded as part of the Multichoice Research Chair in Machine Learning at the University of Pretoria, South Africa. This work is based on the research supported wholly/in part by the National Research Foundation of South Africa (Grant Numbers 46712). Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF. This work is run on the Lengau Cluster of the Center for High Performance Computing, South Africa.

References

1. Adriaensen, S., Brys, T., Nowé, A.: Fair-share ils: a simple state-of-the-art iterated local search hyperheuristic. In: Proceedings of the 2014 annual conference on genetic and evolutionary computation. pp. 1303–1310 (2014)

2. Alvarez-Benitez, J.E., Everson, R.M., Fieldsend, J.E.: A mopso algorithm based exclusively on pareto dominance concepts. In: International conference on evolutionary multi-criterion optimization. pp. 459–473. Springer (2005)
3. Chan, C.Y., Xue, F., Ip, W., Cheung, C.F.: A hyper-heuristic inspired by pearl hunting. In: International Conference on Learning and Intelligent Optimization. pp. 349–353. Springer (2012)
4. Drake, J.H., Kheiri, A., Özcan, E., Burke, E.K.: Recent advances in selection hyper-heuristics. European Journal of Operational Research **285**(2), 405–428 (2020)
5. E, B., M, G., M, H., G, K., B, M., G, O., A, P., S, P.: The cross-domain heuristic search challenge–an international research competition. In: International Conference on Learning and Intelligent Optimization. pp. 631–634. Springer (2011)
6. Gutierrez-Rodríguez, A.E., Ortiz-Bayliss, J.C., Rosales-Pérez, A., Amaya-Contreras, I.M., Conant-Pablos, S.E., Terashima-Marín, H., Coello, C.A.C.: Applying automatic heuristic-filtering to improve hyper-heuristic performance. In: 2017 IEEE Congress on Evolutionary Computation (CEC). pp. 2638–2644. IEEE (2017)
7. Hassan, A., Pillay, N.: Java library for dynamic heuristic set selection (September 2021), https://github.com/Al-Madina/Dynamic-Heuristic-Sets
8. Hsiao, P.C., Chiang, T.C., Fu, L.C.: A vns-based hyper-heuristic with adaptive computational budget of local search. In: 2012 IEEE congress on evolutionary computation. pp. 1–8. IEEE (2012)
9. Hyde, M., Ochoa, G., Vázquez-Rodríguez, J.A., Curtois, T.: A hyflex module for the max-sat problem. University of Nottingham, Tech. Rep pp. 3–6 (2011)
10. Meignan, D.: An evolutionary programming hyper-heuristic with co-evolution for chesc11. In: The 53rd Annual Conference of the UK Operational Research Society (OR53). vol. 3 (2011)
11. Mısır, M., Verbeeck, K., De Causmaecker, P., Berghe, G.V.: The effect of the set of low-level heuristics on the performance of selection hyper-heuristics. In: International Conference on Parallel Problem Solving from Nature. pp. 408–417. Springer (2012)
12. Mısır, M., Verbeeck, K., De Causmaecker, P., Berghe, G.V.: An intelligent hyper-heuristic framework for chesc 2011. In: International Conference on Learning and Intelligent Optimization. pp. 461–466. Springer (2012)
13. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., et al.: Hyflex: A benchmark framework for cross-domain heuristic search. In: European Conference on Evolutionary Computation in Combinatorial Optimization. pp. 136–147. Springer (2012)
14. Pillay, N., Qu, R.: Hyper-Heuristics: Theory and Applications. Natural Computing Series, Springer International Publishing (2018)
15. Pillay, N.: A review of hyper-heuristics for educational timetabling. Annals of Operations Research **239**(1), 3–38 (2016)
16. Soria-Alcaraz, J.A., Ochoa, G., Sotelo-Figeroa, M.A., Burke, E.K.: A methodology for determining an effective subset of heuristics in selection hyper-heuristics. European Journal of Operational Research **260**(3), 972–983 (2017)
17. Vázquez-Rodrıguez, J.A., Ochoa, G., Curtois, T., Hyde, M.: A hyflex module for the permutation flow shop problem. School of Computer Science, University of Nottingham, Tech. Rep (2009)