A Hyper-heuristic based on Random Gradient, Greedy and Dominance

Ender Özcan and Ahmed Kheiri

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

Abstract. Hyper-heuristics have emerged as effective general methodologies that are motivated by the goal of building or selecting heuristics automatically to solve a range of hard computational search problems with less development cost. HyFlex is a publicly available hyper-heuristic tool for rapid development and research which provides an interface to four problem domains along with relevant low level heuristics. A multi-stage hyper-heuristic based on random gradient and greedy with dominance heuristic selection methods is introduced in this study. This hyper-heuristic is implemented as an extension to HyFlex. The empirical results show that our approach performs better than some previously proposed hyper-heuristics over the given problem domains.

1 Introduction

Hyper-heuristics represent a class of search methodologies which explore the space of heuristics rather than the space of solutions, directly. There are two main types of hyper-heuristic methodologies in literature: methodologies to *select* or *generate* heuristics [1]. The main goal in this line of research is to raise the level of generality by providing hyper-heuristic solution methodologies that are applicable to different problem domains without requiring any additional development cost. A selection hyper-heuristic is a high level problem solving framework that can select and apply an appropriate low-level heuristic at each decision point, given a particular problem instance and a number of low-level heuristics. This study focusses on the selection hyper-heuristics.

Cowling et al. [2] presented a variety of selection hyper-heuristics embedding simple heuristic selection methods, such as *Random Descent* (Gradient), *Greedy* and a sophisticated one, namely *Choice Function*. Random Descent selects a low level heuristic randomly and applies it as long as the solution is improved. If the solution worsens, then another low level heuristic is selected and the same process is repeated. Greedy applies all low level heuristics to the candidate solution and selects a heuristic, hence a solution which provides the best quality. The new solution could be still worse than the current solution. The authors reported the success of a learning hyper-heuristic; namely, Choice Function. Greedy also showed some potential. The *peckish* heuristic selection methods

attempt to reduce the number of low level heuristics either using an online or an offline mechanism. Cowling et al. [3] suggested a Tabu Search based hyper-heuristic that utilised a list to disallow the use of low level heuristics generating worsening results. More on hyper-heuristics can be found in [1].

To the best knowledge of authors, there are two publically available hyper-heuristic tools: Hyperion [4] and Hyflex [5]. Hyperion provides a general recursive framework for the development of hyper-heuristics (or metaheuristics), supporting the selection hyper-heuristic frameworks provided in [6]. Hyflex provides reusable hyper-heuristic (meta-heuristic) components, having a support for the problem domains of Boolean Satisfiability (MAX-SAT), One Dimensional Bin Packing, Permutation Flow Shop (PFS) and Personnel Scheduling (PS) each with ten different instances and a set of low-level heuristics. Burke et al. [7] investigated the performance of a range of selection hyper-heuristics implemented as part of HyFlex. This was a proof of concept study for CHeSC: Cross-Domain Heuristic Search Competition [1]. The best selection hyper-heuristic will be determined among CHeSC competitors which generalises the best across a set of problem instances from different problem domains. Burke et al. [7] reported that the best performing hyper-heuristic was an iterated local search approach.

This study describes a multistage selection hyper-heuristic. The proposed hyper-heuristic is implemented based on HyFlex. Either Greedy or Random Gradient heuristic selection method is used as a heuristic selection method at any stage. Therefore, each stage will be referred to as Greedy or Random Gradient stage depending on the heuristic selection method used. Heuristic selection is followed by a Naïve move acceptance (NV) strategy [7] to decide whether to accept or reject the new solution considering its quality. The performance of the proposed hyper-heuristic is tested over these problem domains and compared against some previous approaches.

2 Methodology

Figure 1 provides the pseudocode of the proposed hyper-heuristic. The multistage selection hyper-heuristic mechanism starts with a Greedy stage. The Greedy heuristic selection method allows all the low level heuristics to process a given candidate solution successively for a number of steps to build a *List of Active Heuristics* (LAH) in the Greedy stage. LAH is a list of the low level heuristics that are expected to perform relatively well. This is an opposite strategy employed by the Tabu Search based hyper-heuristic [3] which utilises a list to disallow the use of low level heuristics generating worsening results. In the first step of the Greedy stage, LAH contains all low level heuristics. The Greedy heuristic selection method combined with a dominance based strategy is used to reduce the number of active heuristics for the next stage. The Greedy stage is always followed by a Random Gradient stage. The best solution found during the Greedy stage is used as the current solution to be processed by the Random Gradient stage. In this stage, Random Gradient heuristic selection method

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

picks a low level heuristic from LAH randomly and applies it to the solution in hand repeatedly until there is no improvement. In the case of obtaining a non-improving solution, the hyper-heuristic will go into the Random Gradient stage again without accepting the new solution with a probability of P_s; or it will go into the Greedy stage for updating the list of active heuristics with a probability of P_u; or it will accept the non-improving solution with a probability of $(1-P_s-P_u)$ and continue with the Random Gradient stage. The following parts explain how the stages interact in more details.

Framework(Parameter: $S_{initial}$**;** P_s **and** $P_u \in [0,1]$**)**

$S = S_{initial}$; $f = Evaluate(S)$; $L = BuildLAH()$; $LLH = SelectRandomlyFrom(L)$
while $(TimeLimitNotReached)$
  $S' = ApplyHeuristic(LLH, S)$
  $f' = Evaluate(S')$
  if (f' **does not improve** f) **then**
    $r = generateUniformRandomNumberIn(0,1)$
    if $(r < P_s)$
      $LLH = SelectRandomlyFrom(L)$; $P_a = 0$
    else if $(r < P_s + P_u)$
      $L = UpdateLAH()$; $LLH = SelectRandomlyFrom(L)$; $P_a = 0$
    else P_a**=1**
  $S = NaiveMoveAcceptance(P_a, S, S')$
return $\{S\}$

Fig. 1. Pseudocode of the dominance based selection hyper-heuristic.

In the Greedy stage, the Greedy heuristic selection method is employed for n successive steps. The best performing heuristics are determined using a strategy inspired from the concept of Pareto Front [8] in multi-objective optimisation. Given a set of k low level heuristic points LLH=$\{LLH_1, LLH_2, \ldots, LLH_k\}$ in 2-dimensional space, each represented by its x (Step) and y (Fitness) coordinates. At each step, the fitness of each solution generated by the corresponding low level heuristic is calculated. Well performing low level heuristics that have the potential to improve within the n steps are those points that are not dominated by any other point. A low level heuristic may make a small improvement in the solution taking a short time and performance-wise this is as good as a heuristic which improves a solution more taking a longer time. Assuming a minimisation problem where we are seeking for the low level heuristics that generate minimum fitness, a point LLH_i is considered to be dominated by point LLH_j if and only if $(LLH_{i.x} \geq LLH_{j.x})$ and $(LLH_{i.y} \geq LLH_{j.y})$.

Figure 2 shows an example on how to build the list of active heuristics for (k=5) low level heuristics. Note that in the Figure, LLH_2 has been added three times to the list and that makes this heuristic to be selected with higher prob-

ability. Note also that in Step1, LLH_2 and LLH_3 have been both added to the list, since they have the same fitness value and they are not dominated by any other point.

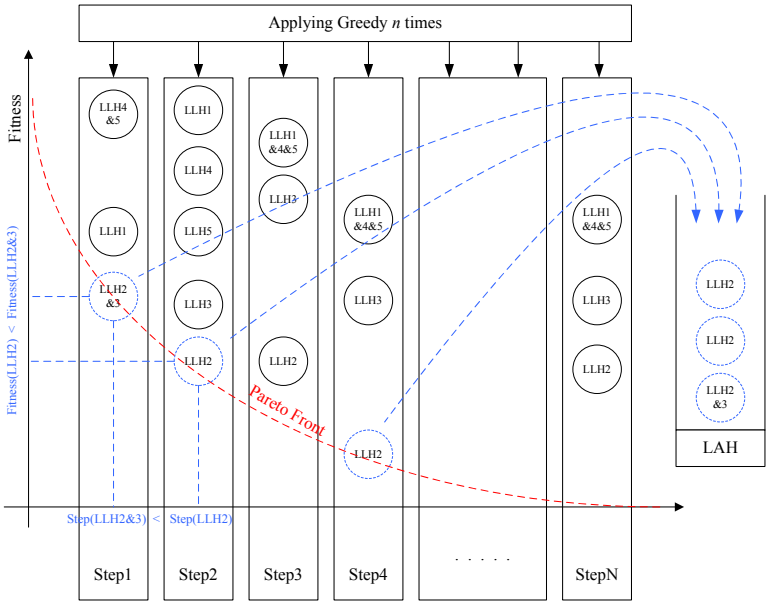

Fig. 2. An illustration showing how the list of active heuristics is built.

As there is a limited time to find the best fitness, the value of n depends on the time of applying Greedy method in one step. n decreases when the time needed to apply the Greedy method was high. An exponential function has been considered to find n, where zero is a possible value. However, in case of n equals zero, then the list of active heuristics will contain the whole k low level heuristics. The value of n will be calculated before starting the main loop and it should be an integer value. $n = Ae^{-f(t)}$ where A is the maximum possible value of n. $f(t)$ is the total time of applying Greedy method for one step divided by the limited time that required to complete the whole process; the total time of applying Greedy method for one step equals the summation of the time required to apply each low level heuristic on a given candidate solution: The goal of this stage is to improve the solution at hand as much as possible turning the framework into a hill climber. A low level heuristic is selected randomly from the list of active heuristics, created during the Greedy stage, and applied repeatedly until no improvement is achieved. The Naïve move acceptance [7] is used as the move acceptance strategy which accepts all improving moves. In case of non-improving move ($P_a = 1$), the solution accepted with a probability of $(1-P_s-P_u)$; otherwise, the solution remains unchanged ($P_a = 0$).

3 Empirical Results

The proposed hyper-heuristic performance is compared to the performances of eight different hyper-heuristics (HH1–HH8) as provided at the competition website and in [7]. It is put into a mock competition against these eight hyper-heuristics based on the Formula1 scoring system. The best hyper-heuristic gets 10 points, the second gets 8, and then 6,5,4,3,2,1 and then all the remaining get no point. These points are accumulated as a score for a hyper-heuristic over all instances from four problem domains each with ten instances.

The parameter values are chosen as A=9, P_s=0.50 and P_u=0.25. These values are decided after a set of exhaustive experiments using different combinations of values which is not reported in this paper due to space requirements. A single run is performed using each problem instance. The experiments were performed on an i3 CPU M330 at 2.13GHz with a memory of 4.00GB. A run terminates after 946 seconds as the competition requires. This value is obtained using the benchmarking tool provided at the competition website

In the MAX-SAT problem domain, our hyper-heuristic produces the best result in 3 out of 10 instances and there is a tie in 2 instances. It is the best hyper-heuristic in this domain. In the bin packing problem domain, our hyper-heuristic performs still well, but in the personnel scheduling and permutation flow shop problems, its performance is not as good as expected. It is observed that mostly, hill climbing heuristics are chosen. Table 1 summarises the results for each problem domain. The proposed hyper-heuristic performs better than the previously proposed hyper-heuristics in the Max SAT and 1D Bin Packing problem domains and worse in the rest of the domains. In the overall, our hyper-heuristic ranks the first with a score of 249.0.

Table 1. Comparisons of the different hyper-heuristics over each domain based on Formula1 scores. The best values are highlighted in bold.

Domain	HH1	HH2	HH3	HH4	HH5	HH6	HH7	HH8	PHH
MAX-SAT	37.0	67.5	47.0	35.5	0.0	64.0	42.0	2.0	**85.0**
1D Bin Packing	37.0	38.0	80.0	63.0	0.0	59.0	21.0	0.0	**82.0**
Personnel Scheduling	**63.0**	61.0	14.0	54.0	36.5	0.0	61.0	33.5	57.0
Permutation Flow Shop	33.5	28.0	24.5	74.0	7.0	**84.0**	30.0	74.0	25.0
Overall	170.5	194.5	165.5	226.5	43.5	207.0	154.0	109.5	**249.0**

4 Conclusion

In this study, a multistage selection hyper-heuristic combining two heuristic selection methods and a Naïve move acceptance method is described. Greedy with dominance and Random Gradient are used as the heuristic selection methods in

an alternating manner at successive stages. Greedy aims to detect the low level heuristics with good performance and maintains a list of active heuristics considering the trade-off between the change (improvement) in the solution quality and the number of steps taken. If a heuristic takes a large number of successive steps and generating a large improvement in the solution quality, the performance of this heuristic is considered to be similar to the one which takes less number of successive steps and improves the solution quality less as well. Random Gradient selects from the (possibly) reduced set of low level heuristics to improve the solution in hand at each step. Whenever the search by Random Gradient stagnates, then the Greedy stage may restart for detecting new list of active heuristics. This is a viable strategy considering that at different points during the search, different heuristics may be performing well. The experimental results show that our hyper-heuristic is a general methodology and performs better than eight previously proposed hyper-heuristics based on their overall rankings considering all problem instances from four different domains.

References

1. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. School of Computer Science and Information Technology, University of Nottingham, Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747 (2010)
2. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling, London, UK, Springer-Verlag (2001) 176–190
3. Cowling, P., Chakhlevitch, K.: Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In: Proceedings of the 2003 Congress on Evolutionary Computation. (2003) 1214–1221
4. Swan, J., Özcan, E., Kendall, G.: Hyperion - a recursive hyper-heuristic framework. In Coello, C.A.C., ed.: Learning and Intelligent Optimization, 5th International Conference, LION 5. LNCS (2011)
5. Burke, E., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vazquez-Rodriguez, J.: Hyflex: A flexible framework for the design and analysis of hyper-heuristics. In: Proceedings of the Multidisciplinary International Scheduling Conference (MISTA09). (2009) 790–797
6. Özcan, E., Bilgin, B., Korkmaz, E.: A comprehensive analysis of hyper-heuristics. Intelligent Data Analysis (2008) 3–23
7. 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. (2010) 1–8
8. Deb, K.: Multi-objective genetic algorithms: Problem difficulties and construction of test problems. Evolutionary Computation **7** (1999) 205–230