A Greedy Hyper-heuristic in Dynamic Environments

Ender Ozcan
ASAP Research Group,
School of Computer Science,
University of Nottingham,
Jubilee Campus, Wollaton
Road, Nottingham NG8 1BB,
United Kingdom
exo@cs.nott.ac.uk

Sima Etaner Uyar
Istanbul Technical University
Faculty of Electrical and
Electronics,
Computer Engineering
Department,
34469 Maslak Istanbul, Turkey
etaner@cs.itu.edu.tr

Edmund Burke
ASAP Research Group,
School of Computer Science,
University of Nottingham,
Jubilee Campus, Wollaton
Road, Nottingham NG8 1BB,
United Kingdom
ekb@cs.nott.ac.uk

ABSTRACT

If an optimisation algorithm performs a search in an environment that changes over time, it should be able to follow these changes and adapt itself for handling them in order to achieve good results. Different types of dynamics in a changing environment require the use of different approaches. Hyper-heuristics represent a class of methodologies that are high level heuristics performing search over a set of low level heuristics. Due to the generality of hyper-heuristic frameworks, they are expected to be adaptive. Hence, a hyper-heuristic can be used in a dynamic environment to determine the approach to apply, adapting itself accordingly at each change. This study presents an initial investigation of hyper-heuristics in dynamic environments. A greedy hyper-heuristic is tested over a set of benchmark functions.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods

General Terms

Algorithms

Keywords

Hyper-heuristics, dynamic environments, adaptive, greedy, mutational heuristics, hill climbing

1. INTRODUCTION

Many real-world optimisation problems are dynamic in nature, i.e. the environment changes over time. For an optimisation algorithm to be successful in such an environment, it should be able to follow these changes and adapt itself to such changes. A change in the environment may occur due to a change in the objective function, the constraints or the problem instance. Different dynamic environments exhibit different change properties [1]:

- frequency of change
- severity of change
- predictability of change
- cycle length / cycle accuracy

Different types of dynamics require the use of different approaches to handle the changes. For example, if the change is not very severe and the new optimum remains close to the old one, perturbing the current solution slightly may be a better approach; whereas if the environment changes drastically and the new optimum is far from the old one, a simple random restart may be the best option. The success of the method chosen in a dynamic environment is closely related to the nature of the underlying changes. Many studies exist in literature that analyse the approaches proposed to handle different types of dynamics in the environment. A detailed analysis of dynamic environments can be found in [1, 7, 8, 12, 13].

Hyper-heuristics are high level heuristics which perform search over a set of low level heuristics for solving difficult problems [2, 5]. In literature, two broad hyper-heuristic classes can be identified: heuristics to choose heuristics [10] and heuristics to generate heuristics [3]. Most of the former type of hyper-heuristics process a single candidate solution and decide which low level heuristic(s) will be applied at each iteration during the search. No problem specific information is allowed between two levels in a standard Hyper-heuristic framework as illustrated in Fig. 1 [4]. This raises the level of generality and a hyper-heuristic can be applied to another problem in another domain easily. Cowling et al. [4] present most of the simple methodologies that can be used as a hyper-heuristic. One of those methods is the *Greedy* approach. Greedy applies all low level heuristics separately to the current candidate solution and accepts the one that generates the best solution. In this study, a Greedy based hyper-heuristic is used. At the end of this stage, the resulting solution might be worse than the current solution. A new solution is accepted if it has better or the same quality as the current one.

Ozcan et al. [9, 10] show that the choice of low level heuristics affects the performance of a hyper-heuristic. Perturbative heuristics are divided into two classes: *mutational* and *hill climbing* heuristics. Mutational heuristics modify a

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

given solution randomly ignoring the quality of the resultant solution, while hill climbing heuristics guarantee a solution which has better or the same quality as the input in the expense of returning the same solution. The previous studies show that choosing a mutational heuristic and then applying a single hill climbing heuristic yields better results than using the hill climber within the set of low level heuristics.

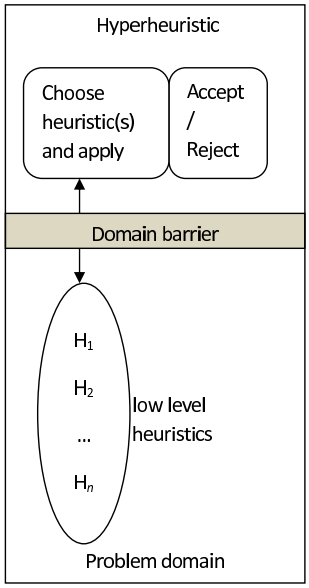

Figure 1: A Hyper-heuristic framework

Due to the generality of hyper-heuristic frameworks, they are expected to be adaptive. Hence, a hyper-heuristic can be used in a dynamic environment to determine the approach to apply, adapting itself accordingly at each change. Since this technique does not require any problem domain specific information, it is applicable, without any modifications, to dynamic environments exhibiting various change dynamics.

The main aim of this study is to perform preliminary experiments to employ a hyper-heuristic framework using mutational heuristics and a hill climber in a dynamic environment while exploring its behaviour in environments exhibiting changes of different severity and frequency. To the best of the authors' knowledge, this is the first study which attempts to use such a Hyper-heuristic framework in the context of dynamic environments.

2. EXPERIMENTS

2.1 Experimental Design

For this preliminary study, we decided to focus on two properties of dynamic environments: frequency and change severity. We also decided to work with mutational heuristics since appropriate diversity levels and convergence rates are among the key factors which determine the success of an algorithm under changes of different severity levels and frequencies. As a basis for the mutational heuristics, the traditional bit-flip mutation in genetic algorithms is adapted. Each bit in a candidate solution is traversed one by one respectively and a bit is filliped with a given probability, referred to as *mutation rate*.

In this study we use a simple hyper-heuristic framework as shown in Fig. 1. Within this framework, we used six heuristics, five of which are mutations with different mutation rates, namely $1/L$, $2/L$, $4/L$, $8/L$, $10/L$ and 0.5, where L is the length of the solution string. The sixth heuristic is a simple hill-climber, the Davis's bit hill climbing method [6]. A random order with respect to the size of a candidate solution is generated and each bit is visited sequentially based on that order. At each step, relevant bit under consideration is flipped. If this change improves the quality of the solution it is accepted, otherwise it is rejected, and then, the next bit is considered. In this framework, if a mutational heuristic is selected then after the mutation step, hill climbing is always invoked. Greedy heuristic selection method is the one that responds to the changing environment through the low level heuristics more rapidly as compared to the others, since all the heuristics are invoked anyway. Hence, it is the one whose performance is investigated in our experiments.

During the experiments five different well known benchmark functions with different characteristics are used: Step, Schwefel, Rastrigin, Salomon and Whitley's 4-bit deceptive function. Whitley's function is a discrete function, while the rest of them are continuous functions. A detailed explanation of these benchmark functions are given in [11]. The settings used for these problems are given in Table 1. In this table, D is the dimension for the problem and n is the number of bits chosen to represent each dimension. For all problems, the solution string is made up of 0 or 1 values. For Whitley's function, the representation is straightforward; each location in the solution string corresponds to one dimension of the function. For the continuous functions, a Gray encoding is used as the representation scheme, where each dimension is represented with a predefined number of bits.

Table 1: Parameter settings for each problem

Problem	D	n	LF	MF	HF
Step	20	10	165	80	30
Schwefel	20	10	155	70	25
Rastrigin	20	10	375	180	70
Salomon	20	10	195	100	40
Whitley	20	4	122750	60000	20000

To experiment with different dynamic environments, four change severity levels and three change frequencies are implemented. For *low severity*, *medium severity*, *high severity* and *very high severity* changes, on average 5%, 20%, 40% and 70% of the bits are changed respectively between consecutive environments. To determine the number of iterations between two consecutive changes which correspond to *low frequency*, *medium frequency* and *high frequency* changes, we let the programs run for a long time without any changes on each problem separately and we observed the convergence plots. The determined settings for the change frequency in terms of number of iterations between consecutive changes are also reported in Table 1. In the table LF, MF and HF correspond to *low frequency*, *medium frequency* and *high frequency respectively.*

In dynamic environments, one of the most commonly used performance comparison metrics is the *offline error* [1]. *Offline error* is calculated as given in Equation 1. We also use this metric to asses performance.

$$x' = \frac{1}{T} \cdot \sum_{t=1}^{T} e'_t \quad and \quad e'_t = max\{e_\tau, e_{\tau+1}, ..., e_t\} \quad (1)$$

where x' is the offline error, T is the total number of evaluations, τ is the last time step ($\tau < t$) when change occurred

2.2 Results

All results are averages over 50 runs for each test. For each run of the algorithm, the run continues until 21 environments are sampled, i.e. 20 changes occur after the initial environment.

To see the effect of using a hyper-heuristic framework, for each environment dynamic and on each problem, we run each heuristic separately as well as the hyper-heuristic framework. We then rank the results of each run based on the *offline error* values averaged over 50 runs. These experimental results are summarised in Table 2.

In Table 2, the first column shows the different change frequency levels as *low*, *medium* and *high*. The corresponding iteration values are different for each problem. Therefore they are not stated explicitly in this table. These can be found in Table 1. The second column shows the different change severity levels where 0.05, 0.2, 0.4 and 0.7 correspond to *low severity*, *medium severity*,*high severity* and *very high severity* changes respectively. $H0$ corresponds to the heuristic which only uses the hill-climber, $H1$, $H2$, $H3$, $H4$, $H5$ and $H6$ correspond to the mutational heuristics which use a mutation rate of $1/L$, $2/L$, $4/L$, $8/L$, $10/L$ and 0.5 respectively. HH corresponds to the hyper-heuristic which uses all six heuristics. Each entry in the table gives the average rank of each tested approach on the five benchmark functions for the corresponding dynamic environment type.

Table 2: Ranking of each approach averaged over five benchmark functions for each dynamic environment type which is determined by a given frequency (f) and severity (α) of change.

f	α	$H0$	$H1$	$H2$	$H3$	$H4$	$H5$	$H6$	HH
low	0.05	6.6	3.8	1.8	1.8	3.8	5.6	7.4	4.8
	0.20	6.8	4.4	2.4	1.6	3.0	4.4	7.4	5.8
	0.40	7.0	4.8	3.0	1.6	2.6	3.8	7.0	5.6
	0.70	7.2	5.0	3.4	1.8	2.4	3.8	7.0	5.4
med	0.05	6.4	4.0	1.8	1.6	3.8	5.6	7.4	4.8
	0.20	6.8	4.8	2.8	1.6	2.8	4.0	7.2	5.6
	0.40	6.6	4.8	3.8	2.0	2.4	3.6	7.2	5.6
	0.70	7.0	5.2	3.8	2.0	2.4	3.2	7.0	5.2
high	0.05	6.2	4.2	2.4	1.6	3.6	5.0	7.4	4.6
	0.20	6.2	5.0	3.6	1.8	2.4	4.0	7.4	5.6
	0.40	6.2	5.6	3.8	2.4	2.0	3.4	7.2	5.2
	0.70	6.8	5.2	4.2	2.6	2.0	2.6	6.8	5.0

When we look at the table, we see that as the change severity increases, the average ranks for those heuristics using higher mutation rates get better. For heuristics with high mutation rates, this effect is emphasised as the frequency increases. For example for H4 and H5, the ranks drop to their 63% and 67% respectively in a low frequency environment, while for the high frequency environment, these values become 55% and 52% respectively. For heuristics with lower mutation rates, the contrary becomes true. For example for H1 and H2, the ranks increase to their 132% and 189% respectively in a low frequency environment, while for the high frequency environment, these values become 124% and 175% respectively. These results are to be expected, because in an environment where the changes are severe, in order to adapt to the new environment, the current solution needs to be perturbed more as opposed to in environments with low severity changes. However, as the changes become more frequent, the algorithm has less iterations between consecutive changes to find the current optimum. Too much diversity slows down convergence, therefore the benefits of increased diversity become less as frequency increases.

For the hyper-heuristic approach (HH) we see that the average ranks mostly lie between 4.8 and 5.6 where the worst rank is 8.0, with no change direction trend as the severity or the frequency increases or decreases. The selection heuristic used in this study is greedy, therefore it applies all heuristics to the current solution to choose the best one as the next solution. To provide a fair comparison between the hyper-heuristic and the others which use a single heuristic, each program is allowed the same number of fitness evaluations between each consecutive change. Note that while the hyper-heuristic applies all heuristics at each iteration, the others only apply one heuristic. This implies that while the hyper-heuristic samples a specific number of new points around the current position to go to its next location (thus making only one move per iteration), the others use those fitness evaluations to sample points iteratively to visit many new locations and move around on the landscape. This explains why the average ranks of the hyper-heuristic are not higher as expected. Keeping the number of iterations (i.e. actual moves) between the different programs fixed, rather than the number of actual fitness evaluations would give different results.

In the extreme cases, i.e. when the mutation rate equals to 0.5 or when there is no mutational heuristic used, performance is the worst for all types of environments. The first one of these two heuristics (H6), denotes that at each step, a new solution is randomly generated. This is equivalent to random walk which is not useful in most problems. The second one (H0), denotes that at each iteration, the solution performs only a hill-climbing step from its current location. This causes the solutions to quickly converge to a local optimum. This is especially detrimental in dynamic environments where the optima change over time. H0 achieves its worst performance when the change severity is the highest and its best performance when the change severity is low and change frequency is high. This is an expected outcome. When the change is not severe, the new optimum is usually not too far from the current one and with the high change frequency rate, the solution has not yet had time to get stuck at a local optimum. Therefore, the algorithm is able to adapt better than in the other change scenarios.

3. CONCLUSION AND FUTURE WORK

This is a preliminary study into the applicability of a very simple hyper-heuristic framework based on a very basic selection and acceptance heuristics. The results confirm that different change dynamics require different approaches to handle the changes and a hyper-heuristic is able to choose an appropriate heuristic at each change instance, thus providing an average performance for all dynamic environments.

The preliminary results are very promising which promote

further study. We will further our experiments to include more complicated selection and acceptance methods which also incorporate some form of learning. Also, we used mutational heuristics to handle the changes. However, there are many approaches which have been shown to be useful in dynamic environments, for example memory-based techniques. We will incorporate more techniques from dynamic environments literature into the hyper-heuristic framework as well as experiment with different frameworks.

4. REFERENCES

[1] J. Branke. *Evolutionary optimisation in Dynamic Environments*. Kluwer, 2001.

[2] E. K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of Metaheuristics*, pages 457–474. Kluwer, 2003.

[3] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward. Exploring Hyper-heuristic methodologies with genetic programming. In *Studies in Computational Intelligence: collaboration, fusion and emergence*, chapter 6. Springer, 2009.

[4] P. Cowling, G. Kendall, and E. Soubeiga. A Hyper-heuristic approach for scheduling a sales summit. In *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000*, Lecture Notes in Computer Science, pages 176–190, Springer, 2000.

[5] P. Cowling, G. Kendall, and E. Soubeiga. Hyper-heuristics: A tool for rapid prototyping in scheduling and optimisation. In *Applications of Evolutionary Computing: Proceeding of Evo Workshops 2002*, volume 2279 of *Lecture Notes in Computer Science*, pages 1–10, Springer, 2002.

[6] L. Davis. Bit climbing, representational bias, and test suite design. In *Proceedings of the 4th Int. Conference on Genetic Algorithms*, pages 18–23, 1991.

[7] Y. Jin and J. Branke. Evolutionary optimisation in uncertain environments – a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

[8] R. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*. Springer, 2004.

[9] E. Ozcan, B. Bilgin, and E. E. Korkmaz. Hill climbers and mutational heuristics in Hyper-heuristics. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, volume 4193 of *Lecture Notes in Computer Science*, pages 202–211, Springer, 2006.

[10] E. Ozcan, B. Bilgin, and E. E. Korkmaz. A comprehensive survey of Hyper-heuristics. *Intelligent Data Analysis*, 12(1):1–21, 2008.

[11] K. V. Price and R. M. Storn and J. A. Lampinen. *Differential Evolution, A Practical Approach to Global optimisation*. Springer, 2005.

[12] K. Weicker, editor. *Evolutionary Algorithms and Dynamic optimisation Problems*. Der Andere Verlag, 2003.

[13] S. Yang, Y.-S. Ong, and Y. Jin, editors. *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Intelligence*. Springer, 2004.