Batched Mode Hyper-heuristics

Shahriar Asta, Ender Özcan, and Andrew J. Parkes*

School of Computer Science, University of Nottingham, Nottingham NG8 1BB, UK
{sba,exo,ajp}@cs.nott.ac.uk,
http://www.cs.nott.ac.uk/∼{sba,exo,ajp}/

Abstract. A common form of a *hyper-heuristic* is a method that controls a search process which uses neighbourhood operators. There have many studies showing that hyper-heuristics are reusable for solving unseen problem instances not only from a particular domain but also different problem domains without requiring any change. However, generally hyper-heuristics have been considered as part of time-contract algorithms, i.e. they are given a fixed execution time, and also used to solve each each instance separately. This paper considers the potential gains and challenges of a hyper-heuristic being able to treat a set of instances as a batch, to be completed within an overall joint execution time, but with the hyper-heuristic free to make its own decision as to how to divide the computational effort between the individual instances. Using the standard CHeSC benchmarks, we show the wide variation in runtimes that occur, and give evidence that this results in a batched mode having the potential for significant gains.

Keywords: combinatorial optimisation, metaheuristics, hyper-heuristics

1 Introduction

A goal of hyper-heuristic [1] research is to raise the level of generality of search methods by providing high level strategies, and associated directly-usable software components, that are useful across different problem domains rather than for a single one. (Note that this general goal is not unique to hyper-heuristics but also occurs in other forms, e.g. in memetic computation [2].) There are two main types of hyper-heuristics depending on whether they do *generation* or *selection* of heuristics [3]. In this paper, we focus on selection hyper-heuristics, and in particular, those that combine heuristic selection and move acceptance processes under a single point search (i.e. not population-based) framework [4]. A candidate solution is improved iteratively by selecting and applying a heuristic (neighbourhood operator) from a set of low level heuristics and then using some acceptance criteria to decide if it should replace the incumbent. We also use the Hyflex (Hyper-heuristics Flexible framework) [5] software tool associated with the CHeSC2011 hyper-heuristic competition[1].

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

∗ Corresponding author.

Usually hyper-heuristics are used to individually and independently solve single instances. However, in some cases, it might well be that a batch of instances need to be solved; where the "batching" simply means that a whole set of instances are to be solved within an overall time limit, but there is no a priori restriction on how much time should be spent on each instance, or even that they need to be treated entirely separately (unlike within the CHeSC2011 competition). A real-world application of this is when many different instances, or maybe many variants of a few instances but with different choices for available resources, need to be solved "as well as possible overnight" so that a decision can be made next day as to which one(s) to use. In this case, it is reasonable to consider that hyper-heuristics should be extended so as to treat the instances collectively as a batch. This batching has two immediate potential advantages:

- "Effort balancing". Better balancing of computational effort across the instances. If some are much easier than others then it seems reasonable that they should be allocated less computational time, and more time allocated to those that will benefit most.
- "Inter-instance learning". If some of the instances are from the same domain (as would be the case in most practical applications) then it makes sense that the hyper-heuristic should be able to learn from the earlier instances in order to perform better on the later instances. This gives an intermediate between online and offline learning.

In this brief paper, we do not consider the interesting challenge of the inter-instance learning. Instead, we provide evidence that there is a significant potential benefits of the better balancing of computational effort between instances. Note that, although we do not here provide a mechanism that would be able to directly exploit the potential, the aim is to show that it would be worthwhile for hyper-heuristics research to develop such effort balancing schemes.

After a brief discussion of Hyflex and the CHeSC competition, we study some statistics of the performance of a particular hyper-heuristic on the competition instances; showing that there is a wide variation in their properties, and that this can lead to about half the computational effort effectively being wasted.

2 Background

Hyflex is an interface supporting development and research of hyper-heuristics and other metaheuristics in which the domain level is separated from the hyper-heuristic level. In order to discriminate between the interface from its implementation, we will refer to its first version as Hyflex v1.0 which was developed at the University of Nottingham by a team of researchers of the ASAP research group during 2009-2011. Hyflex v1.0 was used for the "Cross-domain Heuristic Search Challenge" (CHeSC) in 2011. CHeSC2011 used the following problem domains: Boolean Satisfiability (SAT), One Dimensional Bin Packing (BP), Permutation Flow Shop (FS), Personnel Scheduling (PS), Vehicle Routing Problem (VRP)

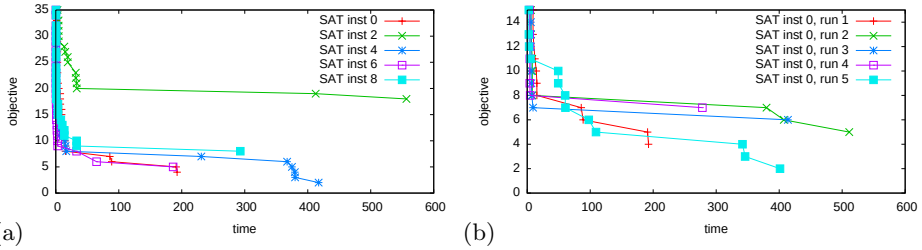


Fig. 1. Performance Profiles for runs on instances from the SAT domain. The Best-So-Far (BSF) quality is plotted against running time; and the lines stop when no further improvements are made within the 600 nsec limit. (a) A single run on each of 5 separate instances. (b) 5 separate runs on a single instance

and the Traveling Salesman Problem (TSP). For each domain ten different instances were provided. A hyper-heuristic was given 10 minutes of execution time per each instance on a specified machine. The winner of CHeSC, *AdapHH* [6] was a learning hyper-heuristic which uses a learning adaptive heuristic selection method in combination with an adaptive iteration limited list-based threshold move accepting method. All the problem domain implementations compatible with Hyflex v1.0 have been serving as benchmarks to test the generality of hyper-heuristics.

3 Performance Properties of the Instances

A standard property of a search is the Performance Profile "PP" or the curve of quality versus time. This is used heavily within the area of anytime reasoning [7] but is also relevant to the case of balancing of computational effort between optimisation tasks. To study the PP on the CHeSC2011 instances we used the winning, and publically-available, hyper-heuristic [6]. Some examples of the PP on the SAT domain are given in Figure 1 and from these we immediately see that those are cases for which improvements in quality cease well before the standard time deadline of 10 (nominal) minutes[2].

This suggests that it could be worth transferring time from such instances to others in which the search does not stagnate. In order to quantify this we performed experiments with the overall deadline extended to 30 (nominal) minutes per instance. For each run, we determined "t(LastImp)" the time at which the last improvement occurred in each run. The results of these are analysed in two ways. Firstly, in Table 3 we compare the fraction of time that is spent

[2] In experiments, the "10 minute" is a "nominal" (or normalised) standardised time as determined by a benchmarking program available via the CHeSC website. On faster machines the real seconds will be smaller; it happens to be 415 secs on the machine we used, however, to aid future comparisons, we always report results using nominal seconds (nsecs) or nominal minutes (nmins)

Table 1. For each domain, and then the aggregate of all domains, we give the average "non-stagnant fraction of the runtime" that was taken to reach the last improvement. (Based on runs of 1800 seconds per instance).

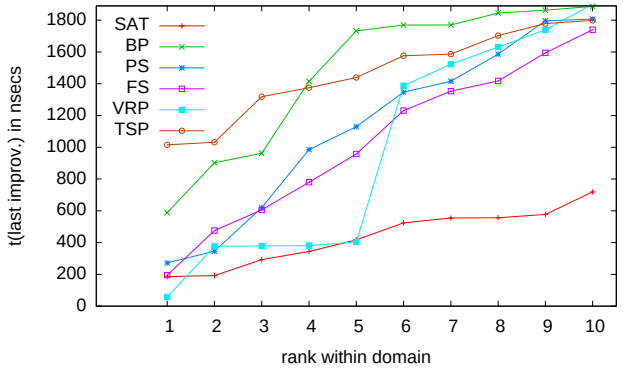
Domain	non-stagnant fraction of runtime
SAT	0.24
BP	0.78
PS	0.62
FS	0.57
VRP	0.52
TSP	0.81
ALL	0.59

before the last improvement against the overall time. We see that, on average, around half the run time is actually "wasted" in the sense that it is after the last improvement.

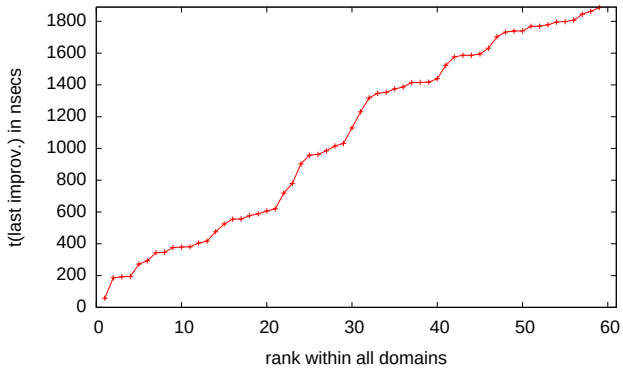
In Figure 2(a) we give the results of ranking the instances in a domain by their value of t(LastImp) and then plotting t(LastImp) against this rank. In the SAT domain we see that most instances stagnate fairly early. In other domains there are a wide range of these stagnation times.

In Figure 2(b) we use the same data, but give the ranking over the domains aggregated together. We see that the instances show widely different behaviours. In particular, around 10 instances stagnate well before the usual 600 nsecs deadline; in contrast, many other instances would potentially benefit from a longer runtime.

The dispersion of stagnation times suggests that each instance should be given a new time limit. There are many potential ways to do this, however, we here we just use a simple scheme. Some instances are selected to be 'down' in that their time limit is reduced to 500 nsecs (from the usual 600 nsecs), and others are up with their time limit extended to 700 nsecs. (The choice of 500 nsecs is fairly arbitrary, and is just to provide an example). Equal numbers of up and down are taken so that the average is still 600 nsecs per instance. We selected the up and down instances based on the inspection of trial runs. We then tested the effects of the new time limits using a different set test runs. Specifically, using 3 trial runs, 9 instances were selected to be down and 9 up. The test runs needed to compare the qualities achieved at 500 vs. 600 seconds for the down instances, and 600 vs. 700 for the up instances. For efficiency, we use the "Retrospective Parameter Variation" technique in the style of [8] (originally used for the loosely-related topic of restarts) so as to re-use data from long runs to quickly simulate shorter runs of different lengths A run on a down instance is called a loser if it improved between 500 and 600, as the reduced time would have cause a loss of quality. A run on a up instance is called a winner if it improved between 600 and 700, as the increased time would have cause an improvement in quality. With 2 runs per instance, then only 1 of the 18 down runs were losers, whereas 15 of the 18 runs on up instances were winners. This roughly corresponds to a net gain of



(a)



(b)

Fig. 2. Times till last improvement on the instances after ranking. (a) For each domain separately, (b) for all the domains together.

improvement on 7 instances, and shows that even simple transfers of runtimes between instances have a potential for significant improvements.

4 Summary and Conclusion.

We proposed a batch mode in which the hyper-heuristic is given many instances and an overall time limit, but is not unrestricted to treating them independently and with the same run-time. We saw the large variation in run times for the CHeSC2011 benchmark instances, and provided a simple change to time limits that lead to significant improvments. Of course, a key question for future work is whether such decisions as to better timing can be made dynamically and in advance, by using the properties of the performance profiles. However, this paper should be taken as indication that if such predictions can be made, then potentially significant runtime can be saved.

We also remark, that it seems that the 6 different CHeSC2011 domains have rather different properties with respect to the standard time limit of 600 nsecs. This wide distribution was presumably good for the competition as it made it more likely that there would be a good differentiation between hyper-heuristics, though it does suggest some caution needs to be taken when interpreting results. It might well be that differences between domains occur because the standard 600 nsec limit occurs at a different phase within the search process; either in the initial "easy" improvements or during the later "harder" stages where improvements are harder to find. Future analyses might benefit from longer run-times to classify the time limit with respect to the expected "stagnation times".

For future work, we intend to extend the Hyflex interface and software to support batched operation, in which the hyper-heuristic is given an overall time limit for the entire set of instances, but is free to make its own decision as to how to partition the computation time between them, and is also able to take what it has learned on one instance and use it for others. This will open up more research opportunities and enable development of automated methods to exploit the power of predicting future performance of a run so as to do better sharing of the computational effort. We would also expect that "online but intra-instance" learning should also lead to many interesting challenges.

References

1. 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
2. Chen, X., Ong, Y.S.: A conceptual modeling of meme complexes in stochastic search. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on **42**(5) (Sept. 2012) 612 –625
3. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. JORS (to appear)
4. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. Intelligent Data Analysis **12**(1) (2008) 3–23
5. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., Parkes, A.J., Petrovic, S., Burke, E.K.: Hyflex: a benchmark framework for cross-domain heuristic search. In: Proceedings of the 12th European conference on Evolutionary Computation in Combinatorial Optimization. EvoCOP'12, Berlin, Heidelberg, Springer-Verlag (2012) 136–147
6. 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,. (August 2011) 374–393
7. Zilberstein, S., Russell, S.J.: Approximate reasoning using anytime algorithms. In Natarajan, S., ed.: Imprecise and Approximate Computation. Kluwer Academic Publishers (1995)
8. Parkes, A.J., Walser, J.P.: Tuning local search for satisfiability testing. In: Proceedings of AAAI 1996. (1996) 356–362