A Software Interface for Supporting the Application of Data Science to Optimisation

Andrew J. Parkes, Ender Özcan, and Daniel Karapetyan

ASAP Research Group
School of Computer Science, University of Nottingham, UK
{ajp, exo, dxk}@cs.nott.ac.uk,

Abstract. Many real world problems can be solved effectively by metaheuristics in combination with neighbourhood search. However, implementing neighbourhood search for a particular problem domain can be time consuming and so it is important to get the most value from it. Hyper-heuristics aim to get such value by using a specific API such as 'HyFlex' to cleanly separate the search control structure from the details of the domain. Here, we discuss various longer-term additions to the HyFlex interface that will allow much richer information exchange, and so enhance learning via data science techniques, but without losing domain independence of the search control.

Keywords: combinatorial optimisation · metaheuristics · data science · machine learning

1 Introduction

Over the last few decades many highly-effective metaheuristic search methods, working on numerous target problem domains, have been developed. They are generally based on neighbourhood improvement search in which a solution is iteratively changed by using moves taken from one or more neighbourhoods. The generation and acceptance/rejection of the moves is generally controlled by a metaheuristic. The neighbourhoods are often quite sophisticated and involve a fairly deep insight into the domain. However, all-too-often the metaheuristics are relatively simple, rather static, and do not exploit the specifics of the interactions between the neighbourhood search operators. Hyper-heuristics are a technique, and a software architecture, that separates the control (the metaheuristic) from the details of the domain and the neighbourhoods [2]. A key aim allowing learning and statistical techniques, 'data science', to be applied to optimisation without them having to be re-implemented separately for every problem domain; essentially giving a plug-and-play version of sophisticated adaptive metaheuristics. The goal is to lift the control from the domain level up to the higher hyper-heuristic level so that data science methods have access to the details of the search process but in a problem domain independent manner. In a sense, this is refactoring of standard algorithms leading to better 'separation of concerns'; search control agents should not know about the domain details.

We discuss the support of this goal by using the 'HyFlex' (*Hyper-heuristics Flexible framework*) interface[1] [4] to separate the hyper-heuristic control from the details of the domain. In the initial limited interface, the hyper-heuristic simply selects neighbourhood moves (the domain-level heuristics) and in return all it learns about the current solution(s) is the objective value. Although such an interface is narrow, one should note that this is sufficient for some well-known meta-heuristics; e.g. standard simulated annealing can be implemented as a simple hyper-heuristic. If data science techniques for optimisation are to become both easy-to-use and still effective, then a drive should be to extend the interfaces (APIs) towards a clean separation but supporting a rich control and information flow. (There are a few existing examples, e.g. [5], that consider a limited broadening of the interface, allowing increased information flow.) The point of this paper, (which is necessarily brief, 'positional', and with only a few key references), is to strengthen and promote the general point that a significantly richer information flow is still consistent with a clean separation between the control and the domain.

The interest in frameworks enabling implementation of general purpose algorithms is growing; for example, Ryser-Welch and Miller [7] provided an overview of some of those frameworks, including Snappy, SATzilla, ParHyFlex, Hyperion and HyFlex. We focus on *selection hyper-heuristics* which mix and control a predefined set of low level heuristics during the search process [2]. Corresponding to these, an initial version of an interface, HyFlex v1.0 was implemented using Java and used in the first Cross-domain Heuristic Search Competition; CHeSC 2011 [4]. HyFlex connects the high level control layer managing a set of low level heuristics via a *domain barrier* but does not allow any problem specific information flow from the domain to the control level. Problem domain implementation details are hidden from the users so that they could focus on the design of the higher level method that will mix and control the low level heuristics and their settings; giving a for researchers, as well as practitioners, to develop new cross-domain solution methods and solve their problems with reduced effort.

The implementation of a metaheuristic is a special case which is supported by HyFlex. The only restriction is that the metaheuristic has to use the operators provided for a problem domain, or the problem domain implementation needs to be extended to include new operators. HyFlex can already be used as a benchmark to evaluate the performance of metaheuristic/hyper-heuristic methods. HyFlex also allows data science techniques and metaheuristics to be employed at the hyper-heuristic level to build, tune or refine hyper-heuristics via analysis (data collection) and execution modes of operation.

The interface was extended to HyFlex v1.1 [1] to enable treating the problem instances collectively as a batch, and was used in the second Cross-domain Heuristic Search Competition; CHeSC 2014. The extension supports balancing of computational effort between instance; if some instances are much "easier" than others then it seems reasonable that they should be allocated less computational time. More importantly, it also allows inter-instance learning: If some of the in-

[1] http://www.hyflex.org/

stances are from the same domain then it makes sense that the hyper-heuristics should be able to learn from the earlier instances in order to perform better on the latter ones. The implementations of HyFlex also provide implementations of multiple problem domains allowing ideas for search control to be tried and tested with much reduced time and effort. Each problem domain includes implementation of a set of low level heuristics (operators), categorised as 'local search' (guarantees a non-worsening solution), 'mutation' (might be worsening), 'ruin-recreate', and 'crossover'. As well as selecting the heuristic, the hyper-heuristic may aslo need to control the heuristics via parameters. For example, local search can be controlled via a 'Depth of Search' parameter, and mutation/ruin-recreate by an 'Intensity of Mutation' parameters.

HyFlex v1.1 also considers the recent developments in the CPUs by supporting multi-core mode of operation, and also allows solution exchange via external memory. In particular, it allows multiple instances of the same solver to be working on the same problem instance, and to share solutions between the instances via a central pool of solutions. Naturally, this means that the system should aim to learn which solutions are most useful, and so it would be helpful for it to have more information about them. (This is part of the motivation for the 'solution features' discussed in the next section.)

2 Future Extensions to HyFlex

Here we discuss future extensions to HyFlex, including support for better annotations, instance/solution features, distance metrics and multi-objective optimisation. In many situations, metaheuristics are run as time contract algorithms, i.e., they terminate after a given time limit. HyFlex v1.0 has full support for this type of operation, returning the final solution and its quality. In certain situations, time limit can be irrelevant or relaxed and running an algorithm on and off, even running different algorithms at any phase might be preferable. This would require a hyper-heuristic (HH) reading from and writing into a file. The next HyFlex version will support saving of a (set of) solution(s) into file(s) and initialising a (set of) solution(s) from a (set of) file(s). Additionally, we will investigate ways of supporting delta/incremental evaluation, enabling fast computation of the objective values (fitness/cost) of a given solution.

In order to give more context, in Figure 1 we give a more refined picture of the kind of structure that often (but not necessarily always) occurs within the hyper-heuristic. Specifically, it can (often) be split into two portions, "reactive" and "reflective". The reactive or 'dispatch' portion of the HH is directly responsible for calling the low level operators; typically, it make such decisions based on some control parameters. The parameters used by the dispatch side are controlled by the reflective portion that 'monitors' the sequence of actions (heuristics selected, etc.) and their effects (changes in the objectives, etc). It uses data science techniques, aiming to dynamically set the control parameters to better values. As an example, the hyper-heuristic might be a form of adaptive reactive simulated annealing, and the parameter could be the temperature. The

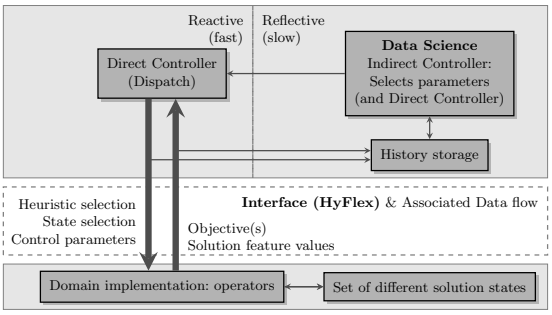

Fig. 1. Proposed general architecture of HyFlex 2.

reflective portion of the HH could then try to observe the search progress in order to decide cooling rates, and make reheats. In this view, a metaheuristic may often be rather static and so could be considered as a hyper-heuristic that lacks the reflective data-science component. The reactive portion bears the responsibility for the 'selection of low-level heuristics' whereas it is the combination of both reactive and reflective that might be said to be closer to doing a 'search the space of meta-heuristics'.

Instance Features: Each problem instance in a given domain carries a set of features reflecting its specific characteristics. There are existing optimisation approaches, such as, algorithm portfolios, that make use of the instance features for choosing the best approach and/or best setting of an algorithm to solve a given instance. Our aim is that HyFlex should support the use of machine learning techniques to relate such characteristics to the choice of hyper-heuristic components or their parameter settings. HyFlex v1.0 does not support instance features; however, the basics for this are already in HyFlex v1.1: In CHeSC 2014, *size* of an instance was provided as an instance feature which can be used for better balancing of computational effort across the instances, i.e. allowing allocation of computational time for the 'smaller' instances. Other instance features could be graph density, number of constraints or planning horizon. Importantly, no domain-specific semantic information will be provided; the hyper-heuristic level will treat the set of feature values features as abstract vectors describing the instance, but will need to reflect and discover for itself how these relate to the search control.

Solution Features: Within domain-specific methods it is quite likely that some features of the current solution would be exploited to guide the search. It seems reasonable that in many cases these could be exposed to the hyper-heuristic as (for example) a vector of values. The meaning of the values would not be known to the hyper-heuristic, however, it could still extract information about the patterns that occur and use that in order to guide the search; e.g. spotting correlations between the solution features and the true objective(s). If the objective is uninformative (due to plateaux) or expensive to compute, then some

solution features could be used as cheaper surrogates or used to guide the search. A more advanced extension of HyFlex might also all the hyper-heuristic to control the individual heuristics in a fashion that accounts for the solution features — e.g. the hyper-heuristic could supply the selected move operators with some bounds or preferences/goals on the values of the mix of multiple objective and solution features. One specific form of this would be for the operators to internally be optimising a weighted sum, but allowing the hyper-heuristic to control the weights. If the exposed solution features are regarded as a chromosome then one can potentially link with the mixed black-white box concepts [6], and so the hyper-heuristics become closer to the realm of evolutionary computation — and lead towards combining evolutionary/genetic algorithms with other metaheuristics. For example, solution features might have the potential to permit search control that captures the essence of EDA (estimation of distribution) algorithms.

Distances: An natural and straightforward extension to HyFlex is to allow the domain level to provide some measure of the difference between different solutions. This is algorithmically useful, for example, to support methods to maintain diversity within populations. This extension has been considered previously [5] for the simplest case of a single distance metric. However, there is no reason not to also permit multiple distance metrics. An annotation system can also say what kinds of properties they satisfy (such as triangle inequality). Note that this does not break the domain barrier as the actual nature of the solutions and the precise meaning of the metric itself still remains hidden. The task of data science would then be to extract useful information so as to control the search.

Multi-objectivity: Another natural and obvious extension to HyFlex is that it should allow the hyper-heuristic access to multiple objectives rather than a single one (e.g. see [3]). There are already studies on approaches that mix and control multi-objective evolutionary algorithms, which is currently not possible with HyFlex. The support for multiple objectives should then enable implementation of evolutionary search methods within the HyFlex and hyper-heuristic context. The primary difference would be that the details of the mutations (or perturbations) are implemented in the domain level, and not visible to the hyper-heuristic. Usually, techniques for diversity are done at the phenotype level (i.e. the objective values), but equivalents of genotypic diversity could be done by also using the distances discussed earlier. Since the hyper-heuristic will have access to the distance metrics, multiple objectives and multiple solutions, then it can measure the quality of the Pareto front, and control the search accordingly.

Annotations: Currently, HyFlex annotates low level heuristics with labels 'mutation', 'local search', 'ruin-recreate' and 'crossover', but an extended typology and annotation system should allow improved implementation of techniques such as iterated local search or memetic algorithms. For example, crossover operators could be annotated by whether they act as 'local search'; whether or not they never generate worsening solutions. Similarly, a ruin and recreate operator could act as a mutational operator in a given domain and as a local search operator in another domain. Furthermore, annotations can be extended to cover instance/solutions features, objectives and distance metrics.

3 Conclusion

The crucial conclusion is that HyFlex domain barrier can be modified to permit a much richer search control and information flow, but without losing the essential advantage of the designer of a hyper-heuristic still not needing to become an expert in the specific domain, but instead be able to apply and exploit data science techniques. An obvious task is to continue with the work in [1] in order to implement this and provide appropriate implemented domain solvers. The advantage, and major challenge, for those studying the application of data science methods to optimisation, is then to find techniques to exploit the rich streams of data that will result during runs of the solvers. We believe that the popular metaheuristics of today barely scratch the surface of what is possible in such a system. We intend to continue with the extensions to HyFlex, including initially support for initialisation from a solution file, saving of a solution, better annotations, instance and solution features, distance metrics, multi-objectivity. In order to reach a wider audience/users, training and teaching material will also be provided. Finally, we remark that using learning at the hyper-heuristic level does not exclude also learning at the domain level; though, expect this adds the challenge of the use of learning to control systems whose behaviour is itself changing due to their own internal learning.

References

1. Asta, S., Özcan, E., Parkes, A.J.: Batched mode hyper-heuristics. In: Nicosia, G., Pardalos, P. (eds.) Learning and Intelligent Optimization, pp. 404–409. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2013)
2. 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. Journal of the Operational Research Society 64(12), 1695–1724 (2013)
3. Maashi, M., Özcan, E., Kendall, G.: A multi-objective hyper-heuristic based on choice function. Expert Systems with Applications 41(9), 4475–4493 (2014)
4. 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: Evolutionary Computation in Combinatorial Optimization, LNCS, vol. 7245, pp. 136–147 (2012)
5. Ochoa, G., Walker, J.D., Hyde, M.R., Curtois, T.: Adaptive evolutionary algorithms and extensions to the HyFlex hyper-heuristic framework. In: Proceedings of the Parallel Problem Solving from Nature - PPSN. pp. 418–427 (2012)
6. Parkes, A.J.: Combined blackbox and algebraic architecture (CBRA). In: Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT '10). pp. 535–538 (2010)
7. Ryser-Welch, P., Miller, J.F.: A review of hyper-heuristic frameworks. In: Proceedings of the Evo20 Workshop, AISB 2014 (2014)