Literature review for GAPSO: development of hyper-heuristcs and population initialization methods

Michał Okulewicz

December 3, 2022

1 Introduction

With the rise of multiple biologically inspired optimization methods in the recent years, the area of designing truly new sampling search strategies seems to be largely explored, at least for now.

Therefore, it would seem that a more promising area of research is a better management of existing techniques, rather than search for an entirely new one. This literature review is aimed at assessing state of the meta-optimization in two areas: hyper-heuristics and population initialization schemes.

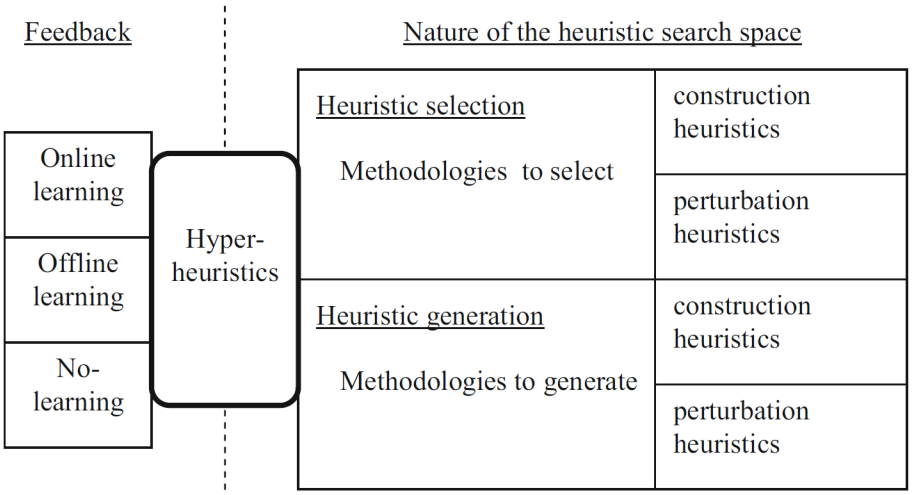

Figure 1: Hyper-heuristics classification reproduced from [4].

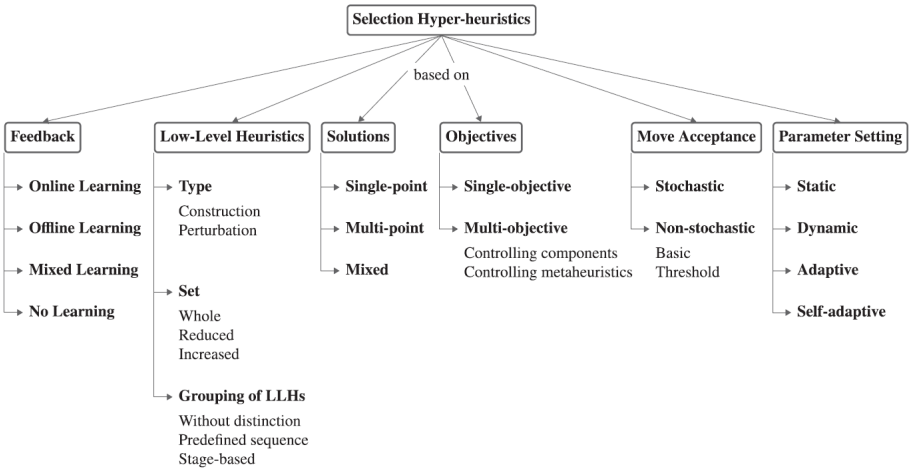

Figure 2: Hyper-heuristics classification reproduced from [9].

2 Hyper-heuristics

The term hyper-heuristics has been introduced by Cowling et al. [6] and further classified and popularized by Burke et al. [2] as "(meta)heuristics working on (meta)heuristics". Generally, Burke et al. [2] promote the view of hyper-heuristics where low level heuristics are treated as a black-box problem in the hope of finding a general purpose high level method of selecting them.

Review [3] from 2013 revisits the state of the hyper-heuristics domain after a period of growing popularity. It acknowledges the fact that while the term is new, the idea itself is not. Burke et al. [3] attribute the origin of research in this area to Fisher and Thompson in 1963 [10], because of their findings that random ensembles of heuristics generate better results than any of them separately. Additionally, they observed the ability to train such ensembles in order to achieve even better performance.

This review [3] acknowledges the fact that Genetic Programming [12, 13] should also be recognized as a type of hyper-heuristic. Finally review [4] from 2019 proposes a classification as presented in Fig. 1. Within this classification the original understanding of term hyper-heuristic is classified as heuristic selection, while methods like Genetic Programming are considered to fall into the category of heuristic generation. Furthermore, each of this classes can be either trained on the problem or act from prescribed recipe, and could be applied to work on construction heuristics or perturbation (search) heuristics.

In the case of applying hyper-heuristics to continuous optimization, the most natural class would be heuristic selection working on perturbation (meta)–heuristics.

Review [9] from 2020 extends the classification of hyper-heuristics (as pre-

sented in Fig. 2). More importantly, it also discusses a few applications of this approach in the domain of continuous optimization [5, 8, 11, 17].

Finally, it is also worth to mention a review of existing hyper-heuristic frameworks from 2014 [16].

2.1 Hyper-heuristics for continuous optimization

Article [17] is an example of work on DE for continuous constrained optimization, where hyper-heuristic uses roulette wheel selection to choose among the 12 basic DE variants for the current iteration. Alternatively, in order to gather data about performance of all the variants of DE, it is selected at random from all possibilities.

Article [11] points out the importance of diversity among the utilized low-level meta-heuristics. It utilizes 4 algorithms as the low-level meta-heuristics: a variant of Particle Swarm Optimization (PSO), a variant of Differential Evolution (DE), a continuous version of Genetic Algorithm (GA) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES). Tabu Search is used as the control algorithm. The study showed that gradually increasing the set of available low-level meta-heuristics resulted in the best performance of the proposed method. While achieving clear advantage over CMA-ES was possible only with the a priory knowledge of the best performing method on a given problem. It should be noted, that this approach is quite similar to GAPSO.

Article [8] utilizes Artificial Bee Colony (ABC) and Krill Herd (KH) as low-level optimizers. The choice of the particular optimizer for the agent is done on the basis of the agent rank in terms of its fitness. The paper considers only an inner comparison between the hybrid and its components, so there is no evidence if state-of-the-art performance has been achieved by this approach as neither of the low-level optimizers is a state-of-the-art method.

Article [5] introduces an approach called hyperSPAM. It consists of CMA-ES as initial search algorithm and S and Rosenbrock (R) algorithms as second phase low-level optimizers during hyper-heuristic search. S algorithm performs axis by axis optimization, while R algorithm estimates the gradient through matrix adaptation. The proposed hyperSPAM method has been tested against CMA-ES on CEC and BBOB benchmarks, resulting in similar overall results. It was enough to utilize S and R algorithms at random in second phase, although their adaptation brought further marginal improvement.

2.2 Other notable hyper-heuristic techniques

Within article [15] a sequence of heuristics is trained with Machine Learning Methods. The search space of heuristics is sampled by a tabu search and the optimal sequence is chosen from the created surrogate model. The model classifies sequences of heuristics as good or bad thus limiting the search space. The heuristics classified as good are subsequently tested on the search space and used to re-train the model.

3 Population initialization

This section presents two approaches to population initialization: one maintaining search diversity, the other simply using a brute-force strategy by enlarging population after each restart of the search process.

Paper [7] discusses a method of combining a Novelty Search approach [14] with generic evolutionary approaches. The article points out two important issues:

1. It is reasonable to use ranking of novelty score and fitness score before blending them $(1 - p) * ranked.fitness + p * ranked.novelty$ for normalization.

2. It is even better to use pure novelty while resetting the whole algorithm in a typical stagnation detection-restart approach.

In order to achieve the second point, an archive of representatives is used to select a starting point. Sample the search space against the archive for the most isolated point. compute σ on the basis of k-nearest neighbours from archive and generate population from this distribution.

Article [1] introduces an IPOP-CMA-ES approach. It is a population size increase applied to a basic version of CMA-ES algorithm. For most multi-modal cases such a simple approach (increase population size by factor 2 after each restart) resulted in significant improvement of algorithm performance, while for uni-modal the algorithm's performance remained roughly the same. The article itself discusses also a stagnation detection strategy, but it seems to be tightly couples with internal working of the CMA-ES itself, and not easily transferable to other methods.

4 GAPSO relation to problems in revised literature

As a first note it is important to observe that while hyper-heuristics methodology is general, its applications to continuous optimization domain seem to be in minority, while a vast amount of work is created for the discrete problems (most noticeably to timetabling and scheduling problems).

In terms of classification proposed by [3], full GAPSO algorithm falls into the category of online learning perturbation selection hyper-heuristics. Without the adaptation mechanism switched on, GAPSO could be classified as no-learning hyper-heuristic.

In GAPSO hyper-heuristic scheme is performed at the level of individual particle, while the learning happens at the level of whole algorithm.

Our findings that even a simple random mix of search strategies (behaviours) is beneficial, are in concordance with the earliest findings in this area of research [10].

The difference between the GAPSO and most orthodox view of hyper-heuristics is the lack of thinking of domain barrier, clearly separating the information about heuristic space and search space. Motivation for research on GAPSO is also on finding the optimal set of behaviors, which interact with one another through the samples each of them gathers.

5 Conclusions for GAPSO development perspectives

- Maybe there is a possibility of adding new behaviors to the list, only when the current set fails to further improve the result?
- Quite a lot of works on hyper-heuristics (e.g. [15]) try to learn the whole sequence of heuristics, maybe this is also a direction worth checking?
- In terms of population initialization, creating an additional sparse samples archive, in order to re-initialize algorithm in the less explored area seems to be an easy extension to test against other initialization approaches.

References

[1] A. Auger and N. Hansen. A Restart CMA Evolution Strategy With Increasing Population Size. In *2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1769–1776. IEEE, 2005.

[2] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In *Handbook of Metaheuristics*, pages 457–474. Kluwer Academic Publishers, Boston, 2003.

[3] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[4] Edmund K. Burke, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. A Classification of Hyper-Heuristic Approaches: Revisited. In *Handbook of Metaheuristics*, pages 453–477. Springer, Cham, 2019.

[5] Fabio Caraffini, Ferrante Neri, and Michael Epitropakis. HyperSPAM : A study on hyper-heuristic coordination strategies in the continuous domain. *Information Sciences*, 477:186–202, 2019.

[6] Peter Cowling, Graham Kendall, and Eric Soubeiga. A Hyperheuristic Approach to Scheduling a Sales Summit. In *Practice and Theory of Automated*

Timetabling III. PATAT 2000. Lecture Notes in Computer Science, pages 176–190. Springer, Berlin, Heidelberg, 2001.

[7] Giuseppe Cuccu, Faustino Gomez, and Tobias Glasmachers. Novelty-based restarts for evolution strategies. In *2011 IEEE Congress of Evolutionary Computation, CEC 2011*, pages 158–163, 2011.

[8] R. Damaševičius and M. Woźniak. State Flipping Based Hyper-Heuristic for Hybridization of Nature Inspired Algorithms. In *ICAISC 2017: Artificial Intelligence and Soft Computing*, pages 337–346, 2017.

[9] John H. Drake, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2):405–428, 2020.

[10] Robert D. Fisher and Gerald L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial scheduling*, pages 225–251. Prentice-Hall, Englewood Cliffs, 1963.

[11] Jacomine Grobler, Andries P Engelbrecht, Graham Kendall, and V S S Yadavalli. Heuristic space diversity control for improved meta-hyper-heuristic performance. *INFORMATION SCIENCES*, 300:49–62, 2015.

[12] John R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In *IJCAI*, pages 768–774, 1989.

[13] John R. Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):251–284, 2010.

[14] Joel Lehman and Kenneth O Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336, 2008.

[15] Jingpeng Li, Edmund K. Burke, and Rong Qu. Integrating neural networks and logistic regression to underpin hyper-heuristic search. *Knowledge-Based Systems*, 24(2):322–330, 2011.

[16] Patricia Ryser-Welch and Julian F. Miller. A Review of Hyper-Heuristic Frameworks. In *Proceedings of the Evo20 Workshop*. AISB, 2014.

[17] José Carlos Villela Tinoco and Carlos A. Coello Coello. hypDE: A Hyper-Heuristic Based on Differential Evolution for Solving Constrained Optimization Problems. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II pp*, pages 267–282. Springer-Verlag, 2013.