SAPIAS Concept: Towards an independent Self-Adaptive Per-Instance Algorithm Selection for Metaheuristics

Mohamed Amine EL MAJDOULI
Conception & Systems Laboratory
Faculty of Science
MOHAMMED V University in Rabat, Morocco
elmajdouli@acm.org

ABSTRACT

Per-Instance Algorithm Selection and Automatic Algorithm Configuration have recently gained important interests. However, these approaches face many limitations. For instance, the performance of these methods is deeply influenced by factors like the accuracy of the underlying prediction model, features space correlation, incomplete performance space for new instances, instances sampling and many others. In this paper, an effort to address such limitations is described. Indeed, we propose a cooperative architecture, labeled as the "SAPIAS" concept, composed of a self-adaptive online Algorithm Selection system and an offline Automatic Algorithm Configuration system, working together in order to deliver the most accurate performance. Additionally, SAPIAS is proposed as a methodic concept that the metaheuristics community might adopt to fill in the gap between theory and practice in the field, by providing for theoreticians the ability to continuously analyze the evolution of the problems characteristics and the behavior of the solving techniques as well as providing a ready to use solving framework for practitioners.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**

KEYWORDS

Per-instance Algorithm Selection, Automatic Algorithm Configuration, Metaheuristics

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3326881

1 INTRODUCTION

The metaheuristics field is an active research field for over 50 years. Many important advances have been achieved in several hard problems including transport, logistics, medicines and many others. However, although the genericity and the flexibility they offer, the public adoption rate of metaheuristics as a branch of artificial intelligence shows that these methods are still far behind to be considered as a ready to use end user product in comparison with neural networks, e.g. deep learning, which is not as old as metaheuristics. Thus, a "why" question is imposed.

- **Investigating the "why" question**

Kendall in [1] has studied this question demonstrating the poor adoption of the industrial / commercial sectors of such solving techniques. One among many reasons the study states is that the underlying benchmarking models are not quite relevant or less descriptive to the real-world problem being solved, leading to several difficulties in successfully applying them in real world at a larger scale. Another possible reason we believe could be also the ignorance of their potential in the optimization field as seen by non-experts. Actually, metaheuristics are pointed out as "untrusted" non-deterministic stochastic procedures rather than "proven" solving tools. The terminologies used by the artificial intelligence community "AIC" and the mathematical optimization community "MOC" supports this where the term "solution of a given problem" is referred as being a "feasible solution" for AIC, while the MOC refers to it as being the "proven optimal solution". However, the issue could be even more organic. Unfortunately, a good part of the research community, driven by an abusive interpretation of the No Free Lunch "NFL" theorem conclusion [2], has "misused" the nature inspiration part which at the same time represents one of the main advantages and central aspects of metaheuristics. This has led to more optimization algorithms than optimized problems, [3] and [4] are two clear examples.

- **Automatic Algorithm Configuration: An emerging approach**

As a remedial to this situation, other approaches are gathering more and more attention recently. One approach is the Automatic Algorithm Configuration "AAC" [5]. Although this approach can be applied broadly to any algorithm by automating its parameter tuning, AAC becomes more interesting when it comes to exploiting the modular aspect of metaheuristics. In fact, most if not all metaheuristics follow a generic modular algorithmic scheme which makes it easier for AAC to tune not only a set of parameters for a given metaheuristic, but also by defining a grammar, it can generate as many variants as possible and can go beyond for a hybridization. Having that said, the main goal of AAC is to find the best overall performing configuration of an algorithm over a set of problem instances. However, the overall

performance metric here is somehow insufficient especially if we target a large-scale adoption as discussed earlier, simply because an overall performance means a compromise of good and less good performance over all instances. This is somehow undesirable for the real-world scenarios as the real-world instances' landscapes are not guaranteed to be drawn from the same sampling the AAC used to determine the best configuration.

- **Per-Instance Algorithm Selection: Another promising way**

Another promising way which coherently uses the NFL theorem conclusion is the Per-instance Algorithm Selection "AS" approach [6]. Indeed, AS tries to identify the best performing algorithm for each new given instance based on a prediction model that binds the instance features to the algorithms' performance space. However, this approach faces many limitations. For example, the performance of a Per-Instance AS system is deeply influenced by factors like the accuracy of the underlying prediction model, feature space correlation or incomplete performance space for new instances. Another example of limitations is the information lack of the real performance of all unpredicted algorithms on a new instance in hand during the prediction decision phase.

- **Cooperation to address limitations**

In this paper, we describe an effort to address such limitations. A cooperative architecture, which we call the SAPIAS concept, composed of a self-adaptive online Algorithm Selection system and an offline Automatic Algorithm Configuration system, working together in order to deliver the most accurate performance. The general concept of a "SAPIAS" architecture is composed of four important cooperating layers. The first two layers are supposed to be implemented for an online execution mode and the last two layers for an offline mode as shown.

The first layer (Prediction layer) defines a Per-Instance Algorithm Selection. This is the classical task where, based on a pre-built prediction model, an algorithm is selected to optimize a new instance. Once the optimization process is achieved, the predicted schedule along with the solution found are passed to the second layer called the Advisor layer.

In the second layer (Advisor Layer), two operations are respectively triggered. The first one is using a control block that implements a low budget procedure, called "Prediction Enhancer", to optimize the new instance. The goal here is not to check the prediction accuracy in the precedent layer, but rather detect if an improvement to the output solution of the predicted algorithm can be performed. the Prediction Enhancer identifies the algorithm(s) which the enhancing improvement strategy(s) improve the solution further. The second operation is then performed by grouping the detected algorithms in a schedule allowing them to contribute to the solving process along with the original algorithm once this latter is predicted again. This list of algorithms is also forwarded to the next layer.

The third layer (AAC Layer) aims to find an algorithm (a metaheuristic) by automatically configuring different components used by the set of algorithms received. For this reason, a Two-Phase Automatic Algorithm Configuration is established. In the first phase, an algorithm configuration is performed at the components level. Indeed, an algorithmic framework is constructed first, then an automatic configuration is performed to find a well performing configuration. Afterwards, if a successful configuration is found, the next phase performs an extensive parameter tuning for the new configuration.

The goal of the fourth layer is to merge the performance set of the new algorithm into the performance space, automatically update the algorithm space and rebinding the outperformed algorithms to the new algorithm so that it becomes ready to use at the online mode instead of the outperformed algorithms. The next subsections provide more details about each of the proposed layers.

The remainder of this paper is as follows. The second chapter of this paper describes the different layers of the SAPIAS concept in more details. The third chapter discusses the research question addressed by SAPIAS, the challenges and the possible workouts that can be investigated. A conclusion and perspectives are given right after.

2 SAPIAS: A SELF ADAPTIVE PER-INSTANCE ALGORITHM SELECTION SYSTEM

2.1 Prediction Layer: Defining the Per-Instance Algorithm Selection

Formally, a per instance AS problem can be defined as follows. Given a set I of problem instances, a distribution D over I, a space of algorithms A, and a performance measure $m: I \times A \rightarrow IR$, the per-instance algorithm selection problem aims to find a mapping $s: I \rightarrow A$ that optimizes the performance measure achieved by running the selected algorithm $s(i)$ for instance i in expectation across instances in I drawn from D.

To build an Algorithm selection system in SAPIAS, the following assumptions on the space of algorithms A of size n are considered:

- An algorithm A_i is implemented using the modular scheme composed of an Initialize module IN_i, an Amelioration module AM_i and a Next Generation Selection module SE_i.
- Modules of an algorithm can be used in a standalone mode. Which means every module can be executed by providing the adequate inputs and without the need to run all of the algorithm's modules.

Using the extracted features from pre-available problem instances along with the performance space drawn from each algorithm performance over all instances, a prediction model is built. Upon the arrival of a new instance I_i, this model will select the algorithm A_i that is likely to better solve this instance based on the features projection results, then communicated to the next layer.

2.2 Control Layer: Setting up the solving process

At this layer, a Control Block is implemented where an internal schedule of algorithms is initiated such that:

$$SKDL(A_{ik}) = \alpha_0 A_0 + \cdots + \alpha_i A_i + \cdots + \alpha_n A_n \quad (1)$$

where k is the number of predictions of the algorithm A_i and

$$\begin{cases} \alpha_i = 1 \\ \alpha_j = 0 \end{cases} \quad \text{for } k = 0.$$

This block mainly controls the execution of the solving process. Indeed, using a fixed time budget α_i initially set to *1*, the predicted algorithm A_i is set to optimize the new instance I_i starting from an initial solution S_{init} (or a set of solutions) which results on a solution S_i.

$$S_i = A_i(S_{init}) \quad (2)$$

$$S_{final} = AM_n \circ AM_{n-1} \circ \ldots \circ AM_1(S_i) \quad (3)$$

Afterwards, the solution S_i is passed through all the Amelioration modules implemented in the algorithm space *A* in the goal of identifying one or more amelioration strategies that can enhance S_i further. In the case where no improvement is detected, S_i is returned as a final solution of the instance I_i. Otherwise, when improvements are detected, S_{final} (the improved solution) is returned and the internal schedule $SKDL(A_{ik})$ is updated where the time budget α_i is decreased and the time budgets for other algorithms where the amelioration modules have contributed to S_{final} are increased.

$$C = \{A_l \,/\, \alpha_l > 0\} \quad (4)$$

The next time A_i is predicted, the optimization process is conducted by the schedule of algorithms *SKDL* rather than A_i alone. Also, the set *C* of active algorithms in *SKDL* is passed to the next offline mode layer where an automatic configuration is conducted.

2.3 AAC Layer: Two-Phase Automatic Algorithm Configuration

An Algorithm Configuration problem can be formally defined as follows. Given a parameterized algorithm *A* with possible parameter settings *P*, a set of training problem instances *I* and a performance metric *m: I x P → IR*. The algorithm configuration problem is to find a parameter configuration *p* that optimizes *m* across the instances in *I*.

Given the set of algorithms *C* received from the previous layer, the first automatic algorithm configuration phase has a solid ground to start building new algorithms by combining the algorithms components to generate an algorithm that can provide similar or better performance than the one recorded by *SKDL* in the online mode (e.g. S_{final}). It is also important to note that the set of training instances *I* is reduced to only the new instance I_i. This is particularly performed for three reasons.

The first is that the current component used for the configuration surely contribute to optimize I_i as they were communicated from *SKDL*, and thus we need to shape them into an algorithm. The second reason is that we are particularly interested to have a working algorithm for this new instance as the predicted algorithm was not able alone to provide the best performance and thus future instances that would be considered closer to I_i will find a better suited solving algorithm. The third reason is concerning the simplicity of the process, as starting a full configuration task including the component level and the numerical level with a vast training set of instances would certainly make the task more complex and time consuming.

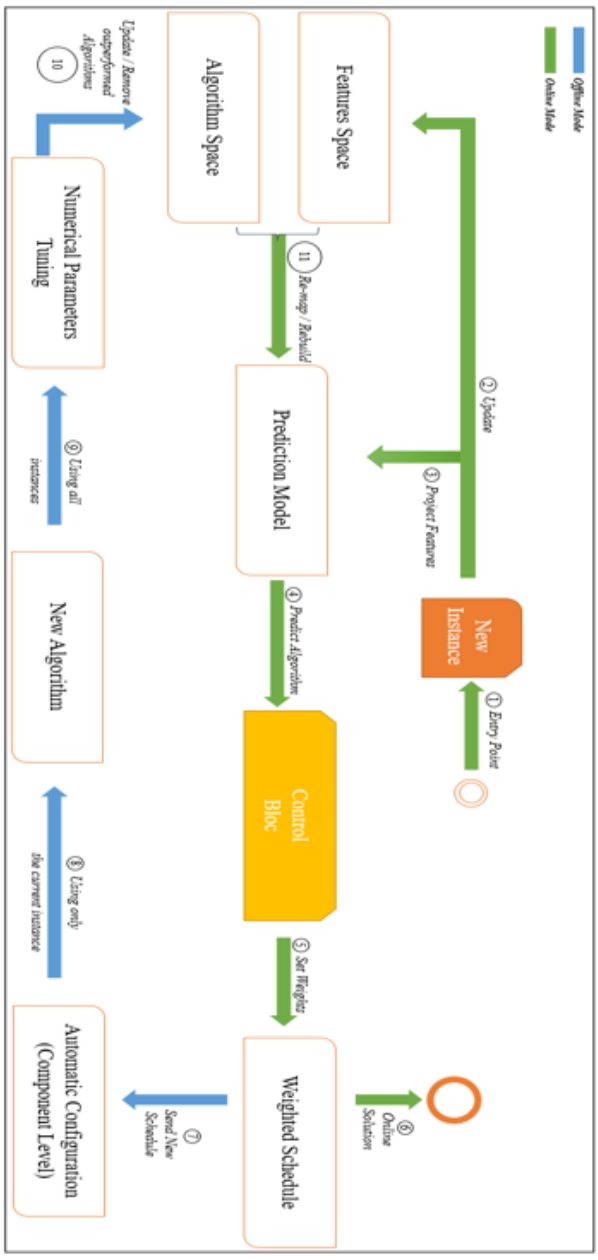

Figure 1 SAPIAS Architecture flow chart

2.4 Update Layer: Online Update of Performance Space

Once the recorded performance set of B is received, the existing performance space is updated with the new algorithm

performance set allowing to remove the binding of algorithms where the new algorithm performs better in their respective instances. Once completed, the algorithm B is made available to the prediction layer immediately at the online mode instead of the replaced algorithms. Consequently, the control block in the second layer is notified in order to reset the schedules.
The prediction model is also updated by regenerating a new one using extracted features from all instances including the newly received instances. This update is ruled using an update threshold monitoring the number of new instances or algorithms being (or waiting to be) made available to the online mode. This allows also to track the variation in the features characterizing the problem at hand rather than being limited to the initial set of features.

3 CHALLENGES & POSSIBLE WORKOUTS

As seen in the previous section, SAPIAS concept tries to deliver an autonomous solving process by exploiting the key benefits in AAC and AS systems. By connecting both, AAC ensures the evolvability of the AS system by providing it with new algorithms for unseen instances, and likewise, AS feeds AAC with a set of algorithms where the configuration procedure is more confident to produce a better algorithm.

However, many challenges can be identified that faces the real implementation of such a concept. For example, the standalone behavior of the "Amelioration modules" in the control block is not guaranteed to be similar to the behavior within the original algorithm. For an evolutionary algorithm, the use of such amelioration blocks is straight forward, however, for a swarm intelligence algorithm, *e.g.* Particle Swam Optimization, this might be complicated as information like local and global best is missing. It is also noticed that the currently designed control block does not make use of the Initialization and the Selection modules for complexity reasons. As a remedial to this situation, a distribution can be learnt from the different values the missing information can take during the training of the algorithm selection system or during the performance space construction. It is also planned in the midterm, to totally replace the amelioration modules in the control block with equivalent highly trusted Estimation Distribution Algorithms that can be used to generate solutions directly. Another issue is the choice of the adopted grammar for the first AAC phase on the component level i.e. *Top–Down*, *Bottom / UP* or use the *Fixed Grammar* of the predicted algorithm. This is an important decision as in case of hybridization of two different strategies like evolutionary and swarm for example, it is easier to integrate the evolutionary process into the swarm intelligence algorithmic scheme than the opposite. Another question that could be solved statistically is how much budget the control block should be allowed. This is very critical in low budget situations where the objective function evaluation is very expensive. This can be solved also by building surrogate models during the performance space construction or the training phase of the Algorithm Selection.

4 CONCLUSIONS & PERSPECTIVES

In this paper, a description of a new concept of using metaheuristics is proposed. This concept, namely SAPIAS, uses a co-evolved Algorithm Selection and Automatic Algorithm Configuration systems in order to deliver the most advanced performance of the solving process. By linking an offline mode AAC system and an online mode AS system using four layers, AS feeds AAC with a set of algorithms where their configuration procedure is more confident to produce a better algorithm. This latter is made available for the AS system to solve more accurately new unseen instances. This design answers some of the limitations known in both systems when used separately. Basically, it is expected that SAPIAS will deliver an independent behavior of the Algorithm Selection where the algorithm space is dynamically updated ensuring an evolvability of the system. Using the amelioration blocks, not only the set of AAC algorithms to be detected is identified but also an enhanced output solution is found. SAPIAS also provides an indirect silent monitoring of the prediction model accuracy by using the schedules strategy, which helps to redress any model fitting problems of the Algorithm selection system. Moreover, a quicker AAC procedure is proposed by dividing the configuration task into two parts. The first one deals with the components' configuration, which upon success, the second part will conduct a numerical parameter tuning.

The SAPIAS concept is intended to be implemented as a community effort and be widespread and ready to use by the industrial sectors in the goal of advancing the evolution of the metaheuristics field in a more concise and beneficial way. By adopting SAPIAS, the research trend that uses the nature inspired aspect of metaheuristics to provide deprecated or renamed models is expected to be widely reduced. It is also expected that the SAPIAS would bring more research directions together to fill in the gap between theory and practice. Actually, theory research can benefit from the analysis of the components' behavior implemented in the control block and link it with the feature space evolution, which is mainly updated using real world instances, in order to propose more accurate surrogate models or evidence-backed parameter values for such components. Alongside with this, practical research can benefit also by exploiting the capability of using and generating as many algorithms variants as possible while being assisted with an underlying learning-based selection system.

REFERENCES

[1] Kendall, G. 2018. Is Evolutionary Computation evolving fast enough? *IEEE Computational Intelligence Magazine,* 13, 2, 42-51.
[2] Ho, Y. C., & Pepyne, D. L. 2002. Simple explanation of the no free lunch theorem of optimization. *Cybernetics and Systems Analysis* 38, 2, 292-298.
[3] Weyland, D. 2015. A critical analysis of the harmony search algorithm—How not to solve sudoku. *Operations Research Perspectives*, 2, 97-105.
[4] Camacho-Villalón, C. L., Dorigo, M., & Stützle, T. 2018, October. Why the Intelligent Water Drops Cannot Be Considered as a Novel Algorithm. In *International Conference on Swarm Intelligence*. Springer, Cham. 302-314.
[5] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated Racing for Automatic Algorithm Configuration. 2016. *Operations Research Perspectives*. 3, 43–58.
[6] Kerschke, P., Hoos, H. H., Neumann, F., & Trautmann, H. (2019). Automated algorithm selection: Survey and perspectives. Evolutionary computation, 27(1), 3-45.