



From Parameter Tuning to Dynamic Heuristic Selection

Yevhenii Semendiak

Yevhenii.Semendiak@tu-dresden.de

Born on: 7th February 1995 in Izyaslav

Course: Distributed Systems Engineering

Matriculation number: 4733680

Matriculation year: 2020

Master Thesis

to achieve the academic degree

Master of Science (M.Sc.)

Supervisors

MSc. Dmytro Pukhkaiev

Dr. Sebastian Götz

Supervising professor

Prof. Dr. rer. nat habil. Uwe Aßmann

Submitted on: 5th April 2020

Aufgabenstellung für die Masterarbeit

Name, Vorname: Semendiak, Yevhenii

Studiengang: Master DSE

Matr. Nr.: 4 7 3 3 6 8 0

Thema:

From Parameter Tuning to Dynamic Heuristic Selection

Zielstellung :

Metaheuristic-based solvers are widely used in solving combinatorial optimization problems. A choice of an underlying metaheuristic is crucial to achieve high quality of the solution and performance. A combination of several metaheuristics in a single hybrid heuristic proved to be a successful design decision. State-of-the-art hybridization approaches consider it as a design time problem, whilst leaving a choice of an optimal heuristics combination and its parameter settings to parameter tuning approaches. The goal of this thesis is to extend a software product line for parameter tuning with dynamic heuristic selection; thus, allowing to adapt heuristics at runtime. The research objective is to investigate whether dynamic selection of an optimization heuristic can positively effect performance and scalability of a metaheuristic-based solver.

For this thesis, the following tasks have to be fulfilled:

- Literature analysis covering closely related work.
- Development of a strategy for online heuristic selection.
- Implementation of the developed strategy.
- Evaluation of the developed approach based on a synthetic benchmark.
- (Optional) Evaluation of the developed approach with a problem of software variant selection and hardware resource allocation.

Betreuer: M.Sc. Dmytro Pukhkaiev, Dr.-Ing. Sebastian Götz

Verantwortlicher Hochschullehrer: Prof. Dr. rer. nat. habil. Uwe Aßmann

Institut: Software- und Multimediatechnik

Beginn am : 01.10.2019

Einzureichen am : 09.03.2020



Unterschrift des verantwortlichen Hochschullehrers

Contents

0.1	abstract	6
1	Introduction	7
1.1	Motivation	7
1.2	Research objective	7
1.3	Solution overview	8
2	Background and Related Work Analysis	9
2.1	Optimization Problems and their Solvers	9
2.1.1	Optimization Problems	10
2.1.2	Optimization Problem Solvers	11
2.2	Heuristic Solvers for Optimization Problems	15
2.2.1	Simple Heuristics	15
2.2.2	Meta-Heuristics	16
2.2.3	Hybrid-Heuristics	20
2.2.4	No Free Lunch Theorem	21
2.2.5	Hyper-Heuristics	22
2.2.6	Conclusion on Heuristic Solvers	25
2.3	Setting Heuristic Parameters	25
2.3.1	Parameter Tuning	26
2.3.2	Systems for Model-Based Parameter Tuning	27
2.3.3	Parameter Control	33
2.4	Combined Algorithm Selection and Hyper-Parameter Tuning Problem	34
2.5	Conclusion	35
3	Concept Description	37
3.1	Search Space	37
3.2	Prediction Process	37
3.3	Low Level Heuristics	37
3.4	Conclusion of concept	38
4	Implementation Details	39
4.1	Hyper-Heuristics Code Base Selection	39
4.1.1	Requirements	39
4.1.2	Parameter Tuning Frameworks	39
4.1.3	Conclusion	39

4.2	Search Space	40
4.2.1	Base Version Description	40
4.2.2	Implementation	40
4.3	Prediction Logic	40
4.3.1	Base Version Description	40
4.3.2	Predictor	40
4.3.3	Prediction Models	40
4.4	Data Preprocessing	40
4.4.1	Heterogeneous Data	40
4.4.2	Base Version Description	40
4.4.3	Wrapper for Scikit-learn Preprocessors	41
4.5	Low Level Heuristics	41
4.5.1	Requirements	41
4.5.2	Code Base Selection	41
4.5.3	Scope of work analysis	41
4.6	Conclusion	41
5	Evaluation	42
5.1	Evaluation Plan	42
5.1.1	Optimization Problems Definition	42
5.1.2	Hyper-Heuristic Settings	42
5.1.3	Selected for Evaluation Hyper-Heuristic Settings	43
5.2	Results Discussion	43
5.2.1	Baseline Evaluation	43
5.2.2	Hyper-Heuristic With Random Switching of Low Level Heuristics	43
5.2.3	Parameter Control	43
5.2.4	Selection Only Hyper-Heuristic	43
5.2.5	Selection Hyper-Heuristic with Parameter Control	43
5.3	Conclusion	44
6	Conclusion	45
7	Future work	46
	Bibliography	47

List of Figures

2.1	Optimization trade-off.	9
2.2	Optimization Target System.	10
2.3	Meta-heuristics Classification.	17
2.4	Evolutionary Algorithm Control Flow	19
2.5	Hyper-Heuristics	22
2.6	Automated Parameter Tuning Approaches.	27

List of Tables

5.1 System settings for benchmark 42

0.1 abstract

Abstract will be available in final versions of thesis.

1 Introduction

Intent and content of chapter. This chapter is an self-descriptive, shorten version of thesis.

1.1 Motivation

Structure:

- optimization problem(OP) → exact or approximate (+description to both) → motivation to use **approximate solvers** →
- impact of parameters, their tuning on solvers → motivation of **parameter control** (for on-line solver) →
- but what if we want to solve a class of problems (CoP) → algorithms performance is different →
- user could not determine it [50] → exploration-exploitation balance
- no-free-lunch (NFL) theorem [88] → motivation of the thesis

thesis motivation The most related research field is Hyper-heuristics optimizations [15], that are designed to intelligently choose the right low level heuristics (LLH) while solving the problem. But the weak side of hyper-heuristics is the luck of parameter tuning of those LLHs [links]. In the other hand, meta-heuristics often utilize parameter control approaches [links], but they do not select among underlying LLHs. The goal of this thesis is to get the best of both worlds - algorithm selection from the hyper-heuristics and parameter control from the meta-heuristics.

1.2 Research objective

Yevhenii: Rename: Problem definition?

The following steps should be completed in order to reach the desired goal:

Analysis of existing studies of algorithm selection. *(find a problem definition, maybe this will do [50])*

Analysis of existing studies in field of parameter control and algorithm configuration problems *(find a problem definition) [55]*

Formulation and development of combined approach for LLH selection and parameter control.

Evaluation of the developed approach with

Yevhenii: family of problems??? since it is a HH, maybe we should think about it...

.

Research Questions At this point we define a Research Questions (RQ) of the Master thesis.

- **RQ 1** Is it possible to select an algorithm and its hyper-parameters while solving an optimization problem *on-line*?
- **RQ 2** What is the gain of selecting and tuning algorithm while solving an optimization problem?
- **RQ 3?** How to solve the problem of algorithm selection and configuration simultaneously?

1.3 Solution overview

Yevhenii: Rename: Problem solution?

- described problems solved by HH, highlight problems of existing HHs(off-line, solving a set of homogeneous problems in parallel)
- create / find portfolio of MHs (Low level Heuristics)
- define a search space as combination of LLH and their hyper-parameters (highlight as a contribution)
- solve a problem on-line selecting LLH and tuning hyper-parameters on the fly. (highlight as a contribution? need to analyze it.)

Thesis structure The description of this thesis is organized as follows. First, in chapter 2 we refresh readers background knowledge in the field of problem solving and heuristics. In this chapter we also define the scope of thesis. Afterwards, in chapter 2 we describe the related work and existing systems in defined scope. In Chapter 4 one will find the concept description of dynamic heuristics selection. Chapter 5 contains more detailed information about approach implementation and embedding it to BRiSE. The evaluation results and analysis could be found in Chapter 6. Finally, Chapter 7 concludes the thesis and Chapter 8 describe the future work.

2 Background and Related Work Analysis

In this chapter we provide reader with a review of the base knowledge in fields of Optimization Problems and approaches for solving them. An experienced reader in field of Optimization and Search Problems could find this chapter as an obvious discussion of well-known facts. If the notions of *Parameter Tuning* and *Parameter Control* are not familiar to you or seems like two different names for one thing, we highly encourage you to spend some time with reading this chapter carefully. In any case, it is worth for everyone to refresh the knowledge with coarse-grained description of sections topics and examine the examples of Hyper-Heuristics in 2.2.5 and Systems for parameter tuning in 2.3.2 since we use them later in concept implementation.

2.1 Optimization Problems and their Solvers

Our life is full of different hard and sometimes contradicting choices.

But what is an Optimization? It is an art of making good decisions.

A decision in working hard or going home earlier, to buy cheaper goods or follow brands, to isolate ourselves or visit friends during the quarantine, to spend more time for planning trip or start it instantly. Each made decisions have it consequences.

Picture 2.1 outlines the trade-off between decision quality and amount of effort spent. Concretely, here you could spend a couple of days searching the interned, comparing routes to find the best suited one, or you could just go. While in former case you seem to waste a lot of time, in later, however, you could lose it in on the way.

Yevhenii: refactor the image

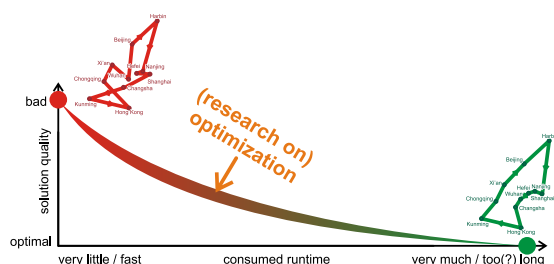


Figure 2.1 Optimization trade-off.

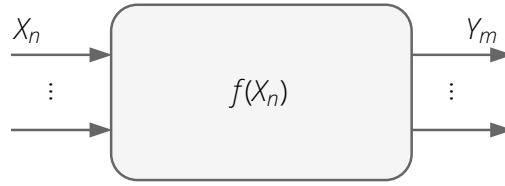


Figure 2.2 Optimization Target System.

2.1.1 Optimization Problems

While the Search Problem (SP) defines the process of finding a possible Solution for the Computation Problem, an Optimization Problem (OP) is the special case of the SP, focused on the process of finding the *best possible* Solution for Computation Problem [37].

In this thesis we focus ourselves on the Optimization Problems.

A lot of performed studies in this field have tried to formalize the concept of OP, but the underlying notion such a vast that it is hard to exclude the application domain from definition. The description of every possible Optimization Problem and all approaches for solving it are out of this thesis scope, but a coarse-grain review should be presented in order to make sure that everyone reading this thesis will be familiar with all used here terms and notions.

In works [2, 9, 32] dedicated to solving OPs, authors distinguished OP characteristics that overlap through each of these works and those we would like to start from them.

First, let us define the Optimization *subject*. Analytically it could be represented as the function $Y = f(X)$ that accepts some input X and reacts on it providing an output Y . Informally it could be imagined as the Target System f (TS) shown on picture 2.2. It accepts the input information with its *inputs* X_n sometimes also called variables or parameters, processes them performing some *Task* and produces the result on its *outputs* Y_m .

Each (unique) pair of sets X_n^i and respective Y_m^i form *Solutionⁱ* for Computational Problem. All possible inputs X^i , where $i = 1 \dots N$ form a *Search Space* of size N , while all outcomes Y^i , where $i = 1 \dots M$ form an *Objective Space* of size M . Note, that both N and M could be equal to ∞ .

The Solution characterized by the *objective* value(s) — a quantitative measure of TS performance that we want to minimize or maximize in Optimization Problems. We could obtain those value(s) directly by reading Y_m , or indirectly for instance, noting the wall clock time TS took to produce the output Y^i for given X^i . The Solution objective value(s) form *object* of Optimization. For the sake of simplicity we here use Y_m , *outputs* or *objectives* interchangeably as well as X_n , *variables* or **parameters**.

Next, let us highlight the Target System characteristics. Among all mentioned in [2, 9, 22, 32] we found those properties the most important:

- **Input data types** of X_m is a crucial characteristic. The variables could be either *discrete* where representatives are binary strings, integer-ordered or categorical data, *continuous* where variables are usually a range of real numbers, or *mixed* as the mixture of previous two cases.
- **Constraints** are functional dependencies that describe the relationships among inputs and define the allowable values for them. As an example, imagine that having X_n equal to *value* implies that X_{n+k} should not appear at all, or could take only some subset of all possible values.
- **Amount of knowledge** that TS exposes about its internals. In other words,

exposed knowledge between the dependencies $X \rightarrow Y$. With respect to this, an Optimization could be: *White Box* — TS exposes its internals fully, it is even possible to derive the algebraic model of TS. *Gray Box* — the amount of exposed knowledge is significant, but not enough to build the algebraic model. *Black Box* — the exposed knowledge is mostly negligible.

- **Randomness** of TS is One of possible challenges, when the output is uncertain. Ideally TS is *deterministic*, when for X^i TS provides the same Y^i all the time. However, in most real-life challenges engineers tackle *stochastic* systems whose output is affected by random processes happened inside TS.
- **Cost of evaluation** is the amount of resources (energy, time, money, etc.) TS should spend to produce the output for particular input. It varies from *cheap* when TS could be an algebraic formula and evaluation Task is simple mathematic computation, to very expensive when the TS is a pharmaceutical company and the Task is to perform a whole bunch of tests for a new drug, which may last years.
- **Number of objectives** is the size of output vector Y_m^i . It could be either *Single* ($m = 1$), or *Multiple* ($m = 2 \dots M$) objective. With respect to the number of objectives, the result of optimization will be either single Solution, or set of non-dominated (Pareto-optimal) Solutions respectively.

Most Optimization Problem types could be obtained by combining different types of each characteristic listed above.

In this thesis we tackle real life combinatorial problem where most prominent incumbents are Bin Packing [60], Job-shop Scheduling [12] or Vehicle Routing [81] Optimization Problems. All combinatorial problems are *NP-Complete* meaning they are both *NP* and *NP-Hard*[34].

As an example, let's grasp these characteristics for Traveling Salesman Problem (TSP) [3] — an instance of vehicle routing and one of the most frequently studied combinatorial OP (here we consider deterministic and symmetric TSP). The informal definition of TSP is as follows: "Given a set of N cities and the distances between each of them, what is the shortest path that visit each city once and return to the origin city?" With respect to our previous definition of an Optimization Problem, Target System here is a function to evaluate the length of proposed path. The TSP distance (or cost) matrix used in this function for evaluation and it is clear that this TS exposes all internal knowledge and those it is a white box. The input X_n is a vector of city indexes those type of input data is non-negative integers. There are two constrains on path: it should contain only unique indexes (those, each city will be visited only once) and it should start and end from the same city: $[2 \rightarrow 1 \rightarrow \dots \rightarrow 2]$. Since the cost matrix is fixed and not changing while during solving process, the TS is considered to be deterministic and cost for two identical paths are always the same (nevertheless, there exist Dynamic TSP where the cost matrix do changes in a runtime to reflect a more realistic real-time traffic updates[18]). It is extremely cheap to compute a cost for given path using cost matrix, those overall Solution evaluation in this OP is cheap, but the overall number of Solutions is $N!$. Since we are optimizing only the route distance, this is a Single objective OP.

2.1.2 Optimization Problem Solvers

Most Optimization Problems could be solved by an exhaustive search, though trying all possible combinations of input variables and choosing one providing the best objective

value. This approach guarantees to find globally optimal solution to OP. But when the Search Space size significantly increase, the amount of time needed for brute-force search becomes infeasible and in many cases even relatively small problem instances could not be solved by an enumeration.

Here different optimization techniques come into play, and exposed by Target System characteristics could restrict and sometimes strictly define the applicable approach. For instance, imagine you have white box deterministic TS with discrete constrained input data and cheap evaluation. The OP in this case could be solved using Integer Linear Programming, or heuristics approaches. If this TS turned out to be a black box, the ILP approaches will not be applicable anymore and one should consider using heuristics [9].

Again, there exist a lot of different characteristics for Optimization Problem Solvers classification, but they are a subject of many surveying works for instance [9]. Here as the point of interest we decided to highlight only two of them.

- **Final solution quality perspective.**
 1. **Exact** Solvers are those algorithms that always provide an optimal Solution for OP.
 2. **Approximate** Solvers produce a sub-optimal output with guarantee in quality (some order of distance to the optimal solution).
 3. **Heuristics** Solvers do not give any worst-case guarantee for the final result quality.
- **Solution availability perspective.**
 1. **Finalizing** algorithms, that report the results only at the end of their run.
 2. **Anytime** algorithms, designed for stepwise solution improvement, while solving the OP and those could expose intermediate results.

Each of these algorithm characteristics provide their own advantages paying sacrificing other. As instance, if solution is not available at any time, one will not be able to control the optimization process, but if it does available, the overall performance may suffer. If the former features are more or less self-explanatory, we explain the later ones in more details including examples.

Solution Quality Classes

Exact Solvers. As stated previously, the exact algorithms are those, which always solve an OP to guaranteed optimality.

For some OP it is possible to develop an effective algorithm that is much faster than the exhaustive search — they run in super-polynomial time instead of exponential still providing an optimal solution. As authors claimed in [87], if the common belief $P \neq NP$ is true, those super-polynomial time algorithms are the best we can hope to get when dealing with NP-complete (combinatorial) problems.

By the definition in [33], the objective of an *exact algorithm* is to perform significantly better (in terms of running time) than exhaustive search. In both works [33, 87] authors enumerated the main techniques for designing exact algorithms, each contributing in this ‘better’ independently, those later they could be combined.

Here is a brief explanation of them:

- **Branching and bounding** techniques when applied to original problem, split the search space of all possible solutions (e.g. exhaustive enumeration) to set of

smaller sub-spaces. More formally this process called *branching the search tree* into sub-trees. This is done with an intent to prove that some sub-spaces never lead to an optimal solution and those could be rejected.

- **Dynamic programming across the sub-sets** techniques in some sort could be combined with the mentioned above branching techniques. After forming sub-trees, the dynamic programming attempts to derive solutions for smaller subsets and later combine them into solutions for larger subsets unless finally original search space solution obtained.
- **Problem preprocessing** could be applied as an initial phase of the solving process. This technique is dependable upon the underlying OP, but when applied properly, significantly reduces the running time. A toy example from [87] elegantly illustrate this technique: imagine problem of finding a pair of two integers x_i and y_i that sum up to integer S in X_k and Y_k sets of unique numbers (k here denotes the size of sets). The exhaustive search will enumerate all $x - y$ pairs in $O(k^2)$ time. But one could first preprocess the data sorting it, after that use bisection search repeatedly in this sorted array and search for k values $S - y_i$. The overall time complexity reduces to $O(k \log(k))$.

Approximate Solvers. When the OP cannot be solved to optimal in polynomial time, people start thinking in alternative ways to tackle it and commonly decide to relax their requirements to optimization. Approximate algorithms are representatives of the theoretical computer science field that have been created in order to tackle the *computationally difficult white box OP* (not solvable in super-polynomial time). Words of Garey and Johnson (computer scientists, authors of *Computers and intractability* book [34]) could play a perfect description of such approaches, built on mathematical proofs: “I can’t find an efficient algorithm, but neither can all of these famous people.”

In contradistinction to exact, approximate algorithms relax the quality requirements and solve an OP effectively with the provable assurances on the result distance from an optimal solution [86]. The worst-case results quality guarantee is crucial in design of approximation algorithms and also involves mathematical proofs.

One may ask a reasonable question “How do these algorithms guarantee on quality, if the optimal solution is unknown ahead?” Certainly it sounds contradicting, since knowing the optimal solution cancels an optimization problem itself. The answer to this question highlights a key technique in the design of approximation algorithms that is out of this thesis scope.

In [86] authors provided several techniques of an approximate solvers design. For instance, the **Linear Programming** relaxation plays a central role in approximate solvers. It is well known that solving the ILP is *NP-hard* problem, however it could be relaxed to polynomial-time solvable Linear Programming. Later fractional solution for the LP will be rounded to obtain a feasible solution for the ILP and different rounding strategies define separate technique for approximate solvers [86]:

- **Deterministic rounding** follows predefined ahead strategy, while
- **Randomized rounding** will perform a round-up of each fractional solution value to integer uniformly.

In contrast to rounding, another technique requires building a *Dual Linear Program* (DLP) for given Linear Program. This approach utilizes a *weak* and *strong duality* properties of DLP to derive the distance of LP solution to an optimal solution of original ILP. Other properties of DLP form a basis for **Primal-dual** algorithms. They start with a

dual feasible solution and use dual information to derive a solution (possibly infeasible) for primal linear program. If the *primal* solution is infeasible, algorithm modifies dual solution to increase the values of the dual objective function. In any case, these approaches are far beyond thesis scope, but in case of interest, reader could start own investigation from [86].

Heuristics. In contradiction to mentioned above solvers, heuristics do not provide any guarantee on the Solution quality. They are applicable not only to white box TS, but also to black box cases. These approaches are sufficient to quickly reach immediate, short-term goal in cases, where finding an optimal solution is impossible or impractical because of the huge search space size.

As in previous approaches, here exist many facets for classification. We will start from the largest one, namely *level of generality*:

- **Simple heuristics** are algorithms, specifically designed to tackle concrete problem. They fully rely on the domain knowledge, obtained from Optimization Problem. Simple heuristics do not provide any mechanisms to escape a local optimum and those could be easily trapped to it [68].
- **Meta-heuristics** are high-level heuristics that not hardly rely on domain knowledge, but also provide some level of generality to control the search. Those, they could be applied to broader range of OPs. Often they are nature-inspired and comprise mechanisms to escape a local optima, but may converge slower than simple heuristics. For more detailed explanation we refer survey [7].
- **Hybrid-heuristics** arise as combinations of two or more meta-heuristics. They could be imagined as recipes merge from the cook book, combining the best decisions to create something new and hopefully better.
- **Hyper-heuristics** is a heuristic that operates on the search space of low-level heuristics choosing or constructing them to solve the OP in hand instead of directly tackling search space of original problem [15].

In upcoming section 2.2, dedicated to heuristics, we discuss each of aforementioned approaches in more details including examples.

The Best Suited Solver Type

"Fast, Cheap or Good? Choose two."

Old engineering slogan.

Now we have reached the crossroad and should make a decision, which way to follow.

First, we have reviewed exact solvers for the Optimization Problems. As mentioned above, they always guarantee to derive an optimal solution. Today, tomorrow, maybe in next century, but eventually exact solver will find it. The only thing we need is to construct an exact algorithm for our problem in hand. This approach definitely provides the best quality of final solution, however it sacrifices simplicity in building a solver and speed in solving the problem.

Next, we have an approximate solver. It does not guarantee to find exactly optimal solution, but instead suggest provably good one. From our perspective, the required effort for constructing an algorithm and proving its preciseness remains the same as

for exact solvers however, this approach beats the previous one in speed of problem solving, sacrificing a reasonably small amount of the result quality. It sounds like a good deal.

Finally, remain heuristic approaches. They are super-fast in comparison to previous two in problem solving, in addition they are much simpler to apply for your specific problem — no need to build complex mathematical models or prove theorems. However, the biggest flaw in these approaches is absence of guarantee in solution quality those, one should consider them up to own risk.

As we mentioned in 2.1.1, in this thesis is dedicated to facing a real-life problems, such as TSP. They showed to be *NP-complete* that is why we are not allowed to apply exact solvers, only approximate and heuristics. In both cases we are sacrificing a solution quality, although in different quantities, but the heuristic algorithms repay in time-to-develop and providing first results. Modern world is highly dynamic, in business survive those who faster and stronger. In most cases former plays the settle role for success. Great products are built iteratively, enhancing existing results step-by-step leaving unlucky decisions behind. It motivates us to stick the heuristic approach within the thesis.

In following section 2.2 we shortly survey different types and examples of heuristics with their properties, weaknesses and ways to deal with them. As the result, we select the best suited heuristics class for solving the problem in hand.

2.2 Heuristic Solvers for Optimization Problems

We base our descriptions of heuristics and their examples on mentioned in section 2.1.1 Traveling Salesman Problem (TSP) [3]. The input data X to our heuristics will be the Target System description in form of distance matrix (or coordinates to build this matrix), while as an output Y from heuristics we expect to obtain the sequence of cities, depicting the route plan.

Most heuristic approaches imply following concepts:

- **Neighborhood** defines the set of Solutions, which could be derived performing single step of heuristic search.
- **Iteration** could be defined as an action (or set of actions) performed over Solution in order to derive a new, hopefully better one.
- **Exploration** (diversification) is the process of discovering previously unvisited and hopefully high quality parts of the search space.
- **Exploitation** (intensification) is the usage of already accumulated knowledge (Solutions) to derive a new one, but similar to existing Solution.

Now we review each mentioned in previous section heuristic type.

Yevhenii: is it needed?..

2.2.1 Simple Heuristics

As we mentioned above, the simple heuristics are domain dependent algorithms, designed to solve a particular problem. They could be defied as rules of thumb, or strategies to utilize exposed by Target System and obtained from previously found solutions information to control a problem-solving process [68].

Scientists draw the inspiration for heuristics creation from all aspects of our being: starting from observations of how humans tackle daily problems using intuition to mechanisms discovered in nature.

Two main types of simple heuristics were outlined in [16]: *constructive* and *perturbative*.

The first type implies heuristics which construct Solutions from its parts stepwise. The prominent example of constructive approach is the *Greedy Algorithm*, a.k.a. *Best Improvement Local Search*. When applied to Traveling Salesman Problem, it tackles the path construction simply accepting the next closest city from currently discovered one. Generally, the Greedy Algorithm follows the logic of making a sequence of locally optimal decisions those ends up in a local optimum after constructing very first Solution.

The second type, also called *Local Search*, implies heuristics which operates on completely created Solutions. The simple example of Local Search is the *Hill Climbing Algorithm*: evaluate entire Neighborhood, move to the best found Solution and repeat. This approach plays a central role in many high-order algorithms, however could be very inefficient, since in some cases Neighborhood could be enormously huge. Another instance of perturbative algorithm is *First Improvement Local Search* [85]. This heuristic accepts better Solution as soon as it finds it during Neighborhood evaluation. The advantage of this methodology over the vanilla Hill Climbing is the velocity of Search Space traversal.

Indeed, since the optimization result is fully defined by the starting point, the use of simple local search heuristics might not lead to a globally optimal solution, but the advantage here is the implementation simplicity [86].

2.2.2 Meta-Heuristics

The meta-heuristic is an algorithm, created to solve wider range of hard optimization problems without need to deeply adapt to each problem.

The research in meta-heuristics field arise even before 1940s, when they have been used already, but not studied formally. In the period of between 1940s and 1980s first formal studies appeared and later till 2000s the field of meta-heuristics arises in wide when numbers of MHs were proposed. The period from 2000 and up until now in [77] authors refer as the time of framework growth. It is now a time, when meta-heuristics widely appear as frameworks, providing a reusable core and requiring only domain specific adaptation.

The prefix *meta* indicates these algorithms to be a *higher level* when compared to simple problem dependent heuristics. A typical meta-heuristic structure could be imagined as $n - T - H$ (template-hook) framework variation pattern. The template part is problem independent and fixed from changes, it forms a core of an algorithm and usually exposes *hyper-parameters* for tuning. The hook parts are domain dependent and those should be adapted for problem in hand. Later T operates by set of Hs to perform an optimization. Many MHs contain stochastic components, which provide abilities to escape from local optimum. However, it also means that the output of meta-heuristic is non-deterministic and it could not provide guarantee on the result preciseness [13].

The success of meta-heuristic optimizer on a given OP depends on the *exploration vs exploitation balance*. If there is a strong bias towards diversification, the solving process could naturally skip a good solution while performing huge steps over the search space, but when the intensification dominating, the process will quickly settle in local optima. Usually it is possible to decompose MH onto simple components and clarify, to which of competing processes contributes each component. Mentioned

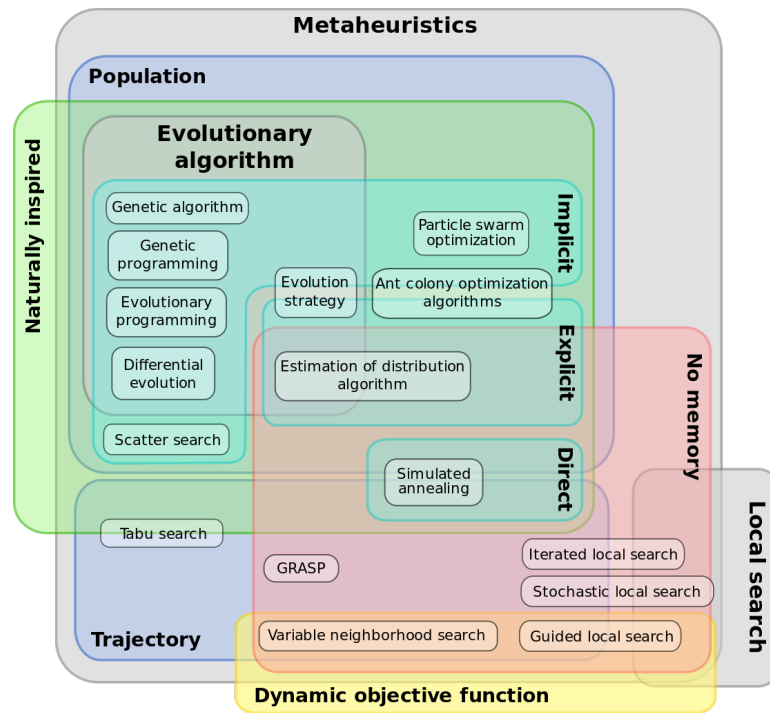


Figure 2.3 Meta-heuristics Classification.

above simple heuristic approaches suffer from high exploitation dominance since they usually do not have components that contribute to exploration.

In general, the difference among existing meta-heuristics lays in a particular way how they are trying to achieve this balance, but the common characteristic is that most all of them are inspired by real-world processes — physics, biology, ethology, evolution, etc.

Meta-Heuristics Classification

The development process of novel methodologies has slowed down and the research community began to organize created algorithms and many classification facets were distinguished.

As an example, authors in [10] classification review highlight following characteristics:

- The **walk-through search space method** could be either trajectory based or discontinuous. The former one corresponds a closed walk on the neighborhood where such prominent examples as *Iterated Local Search* [59] or *Tabu Search* [36] do exist. The later one allows large jumps in the search space with many MH exemplars, such as *Variable Neighborhood Search* [39] and *Simulated Annealing* [51].
- The **number of concurrent solutions** could be either single or multiple (population). Such approaches as Tabu Search, Simulated Annealing or Iterated Local Search are examples of former and Evolutionary Algorithms [27], Ant Colony Optimization [23], Particle Swarm Optimization [49] are instances of later.
- From the **memory usage** perspective, we distinguish those approaches which do and do not utilize memory. The Tabu Search explicitly use memory in forms of tabu lists to guide the search, but the Simulated Annealing is memory-less.
- **Neighborhood structure** could be either static or dynamic. Most local search algorithms, such as Simulated Annealing and Tabu Search are based on static

neighborhood. The Variable Neighborhood Search is an opposite case, where various structures of neighborhood are defined and interchanged while solving an OP.

There are many more classification facets, which are out of this thesis scope. Picture 2.3 illustrates the summarized classification including some skipped characteristics and well-known meta-heuristic instances [19].

Meta-Heuristics Examples

Now we shortly describe some prominent and widely used meta-heuristics. The motivation for doing so comes from later usage of them as the LLH in Hyper-Heuristic, described in section 4.5.

Evolutionary Algorithms (EAs). Evolutionary Algorithms are directly inspired by the processes in nature, described in evolution theory. The common underlying idea in all of these methods is as follows: if we put a population of individuals (Solutions) into an environment with limited resources (population size limit), a competition processes cause natural selection, where only the best individuals survive (compared by the optimization objective) [27].

Three basic actions are defined as operators of EAs: the *recombination* operator that when applied to selected among available in population candidate Solutions (parents) produces new ones (offspring); *mutation* operator applied to Solutions creates a new and very similar one. Applying both operators will create a set of new Solutions — the offspring, whose fitness evaluated on TS. After that, the *selection* operator applied to all available Solutions (parents and offspring) for keeping the population size within defined boundaries. This process is repeatedly iterated until some termination criteria fulfilled, for instance maximal iterations counter reached, number of TS evaluations exceed, or Solution with required quality found. The work-flow of EA depicted on picture 2.4.

Well-known examples of EAs include *Genetic Algorithm* [74], *Genetic/Evolutionary Programming* [54], *Evolution Strategies* [6], and many other algorithms.

Genetic Algorithm (GA) is the first association coming into mind when you hear words 'Evolutionary Algorithms'. GA traditionally has a fixed work-flow: given initial population of μ (we underline hyper-parameters here and in other examples of algorithms) usually randomly sampled individuals, the parent selection operator creates pairs of parents where probability of each solution to become a parent depends on its fitness. After that, the crossover operator is applied to each pair with probability p_c to produce children. Then newly created Solutions undergo mutation operator with independent probability p_m (hyper-parameter). Resulting offspring perform tournament within selection operator and μ survivals replace current population [26]. Distinguishable characteristics of vanilla GA are usage of following operators: bit-strings Solution representation, one-point crossover recombination, bit-flip mutation and generational (only children survive) selection.

Evolution Strategy (ES) algorithms, in contradiction to GA, are working in a vector space of Solution representation and distinguish μ individuals population size and λ offspring generated in one iteration. Often, ES utilizes a very useful feature of *self-adaptation*: changing the mutation step sizes in a runtime. The self-adaptive information is related only to ES, but not to OP under consideration and appended to the individual's chromosome. While the general work-flow for all EAs remains the same, some among underlying operators are changed. Here, parent selection operator take a whole population into consideration, the recombination scheme could involve more

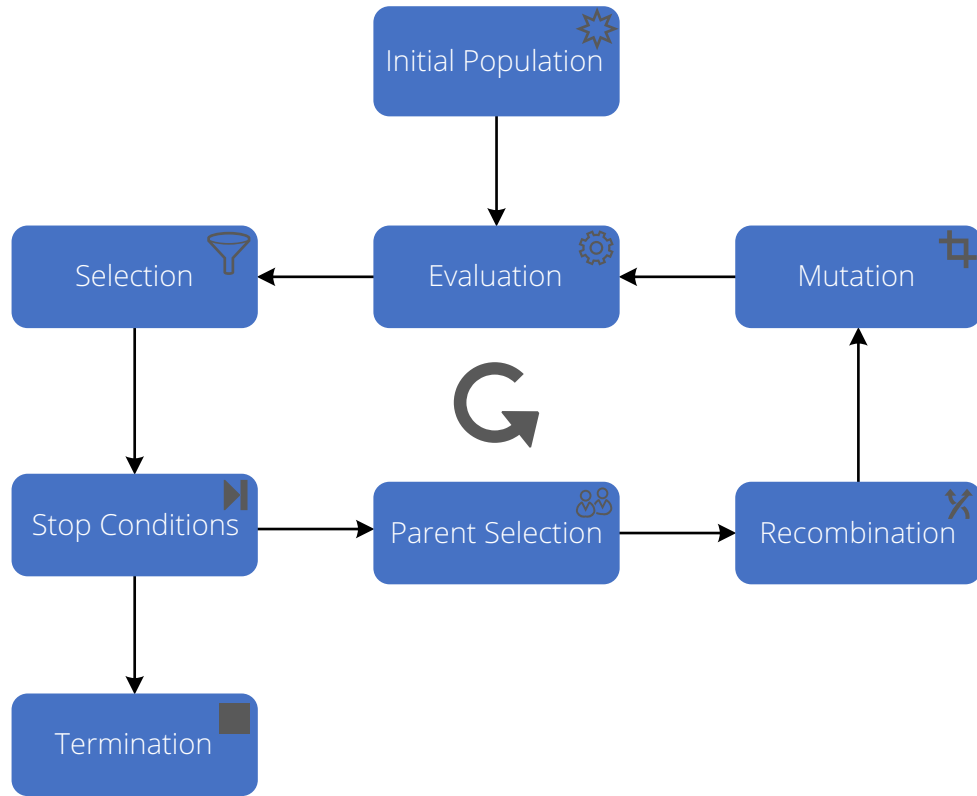


Figure 2.4 Evolutionary Algorithm Control Flow

than two parents to create one child. To construct a child, recombination operator adds alleles from parents in two possible ways: (1) with uniform probability for each parent (discrete recombination), (2) averaging the weights of alleles (intermediate recombination). There are two general selection schemes, used in such algorithms: (μ, λ) — discard all parents and selecting only among offspring highly enriching exploration, and $(\mu + \lambda)$ — include predecessor Solutions into selection, a.k.a. *elitist selection* [26].

Simulated Annealing (SA). This is the other type of meta-heuristic, inspired by the technique used in metallurgy to obtain ‘well-ordered’ solid state of metal. Annealing technique imposes a globally minimal internal energy state and avoids local minimums semi-stable structures.

The search process in SA treated as the metal with a high temperature at the beginning and lowering it to minimum while approaching the end. SA starts with initial Solution S creation (randomly or using some heuristic) and temperature parameter T initialization. At each iteration, new solution candidate S^* sampled within a neighborhood $N(S)$ of current S and the selection criteria evaluated based on S^* quality and T value. S^* replaces S if (1) optimization objective $f(S^*)$ dominates over $f(S)$ or (2) with a probability that depends on quality loose and current T value: $p(T, f(S^*), f(S)) = \exp(-\frac{f(S^*) - f(S)}{T})$. At each iteration the temperature parameter T value is decreased following some type of annealing schedule also called as *cooling rate* [13]. The weak side here is that the quality of each annealing schedule is problem dependent and cannot be determined beforehand, however SA algorithms with parameter control do exist and address this problem by changing the Cooling Rate or temperature parameter T during the search process. Later we shortly review these techniques in section 2.3.3.

2.2.3 Hybrid-Heuristics

The hybridization of different systems often provides a positive effect — taking the advantages of one system you merge them with odds of other system those getting the best from both of them. The same idea applicable in case of meta-heuristics. Imagine you have two algorithms, biased towards exploration and exploitation respectively. Applying them separately, the expecting results in most cases may be far away from optimal as the outcome of disrupted diversification-intensification balance. But when merging them into, say, repeater stages of hybrid heuristic, one will obtain both advantages of escaping local optima and finding good quality results.

Most of available hybridization are done exactly with this idea — *staging combination* of two heuristics, one exploration and second exploitation suited for getting outperforming hybrid.

The methods to construct hybrids are mostly defined by the underlying heuristics and those, to the best of our knowledge, could not be generalized and classified well. One common shared characteristic is usage of *staging approach*, where the output of one algorithm is used as initial state of other.

As for simple heuristics, we will introduce some examples of performed hybridization in order to give you a better understanding of possible hook parts within algorithms and influence of aforementioned balance onto the search process.

Hybrid-Heuristics Examples

Guided Local Search + Fast Local Search [82]. The main focus of *GLS* here is on the Search Space exploration and guidance of process using incubated information. In some sort, *GLS* is closely related to the *Frequency-based memory* used in Tabu Search. During the runtime, *GLS* modifies the cost function of the problem to include penalties and passes this modified cost function to local search procedure. These penalties form memory that characterize a local optimum and guide the process out of it. A local search procedure carried out by the *FLS* algorithm, where the main advantage is quick neighborhood traversal. It is done by braking it up into a number of small sub-neighborhoods, and ignoring those without an improving moves by performing depth first search over these sub-neighborhoods. At some point of time *FLS* reaches the local optimum and passes back the control in *GLS* to repeat the iteration.

Direct Global + Local Search [79]. This hybridization combines mentioned in name optimization strategies into two stages: stochastic global coarse pre-optimization refined by deterministic local fine-optimization. In first stage authors apply one of two described earlier meta-heuristics — Genetic Algorithm, or Simulated Annealing. The transition from Global to Local search happens after reaching the predefined conditions, for instance, when the number of Target System (goal function) evaluations exceeds a boundary, or when no distinguishable improvement was done in the last few iterations. Then, the Pattern Search algorithm [40] also known as direct/, derivative-free/, or black-box search plays role of Local Search heuristic in this combination. The hybrid-heuristic terminates when Pattern Search converges to local optima.

Simulated Annealing + Local Search [61]. After brief explanation of previous two hybrids, an observant reader might make a guess what happens in this particular case, and he will be precisely correct! Authors named this method 'Chained Local Optimization', in other words it is yet another representative of staged hybridization. Iteration starts with the current Solution perturbation — 'kick', referring a dramatic

change of current position within the search space. After this, some Local Search heuristic applied to intensify obtained Solution. Reaching the local optimum, Local Search pass the control flow back to the Simulated Annealing for acceptance criteria evaluation in order to accept or reject a new Solution, which finishes one iteration.

EMILI [67]. Easily Modifiable Iterated Local search Implementation (EMILI) is a framework-like system for automatic generation of new hybrid stochastic Local Search algorithms. *EMILI* is a solver for Permutation Flow-Shop Problems (PFSP), also known as Flow Shop Scheduling problems in which performed a search of an optimal sequence of steps to create products within a workshop. Here authors have implemented both algorithmic- and problem-specific building blocks, defined grammar-based rules for those blocks composition and used an automatic parameter tuning tool *IRACE* [58] to find a high performing algorithm configurations. The work-flow of *EMILI* could be split onto three steps: (1) adaptation of grammar rules to specific PFSP objective representation type (Make-span, Sum completion times, Total tardiness), (2) generation of all possible hybrid heuristics for each of PFSP type and (3) appliance of iterated racing in *IRACE* to select the best performing hybrid for specific problem type.

From our perspective, described approach of automatic algorithm generation is direct incumbent of construction Hyper-Heuristics strategies described in upcoming section 2.2.5, however we are not authorized to change the system class (from hybrid- to hyper-heuristic) defined by *EMILI* authors.

Yevhenii: or we can do it?

2.2.4 No Free Lunch Theorem

A nature question could arise “If we have all this fancy and well-performing heuristics, why should we put an effort in developing new algorithms, instead of using the existing?” And the answer to this question is quite simple — the perfect algorithm suited for all OP does not exist and can not exist. The empirical research has shown that some meta-heuristics perform better with some types of problems, but poorly with others. In addition to that, for different instances of the same problem type, the same algorithm could result in unexpected performance metrics. Moreover, even in different stages of same problem solving process the dominance of one heuristic over another could change.

All search algorithms perform exactly the same, when the results are averaged over all possible Optimization Problems. If an algorithm is gaining the performance in one problems class, it loses in another class. This is a consequence of so-called **No Free Lunch Theorem for Optimization (NFLT)** [88].

In fact, one could not predict, how exactly will behave one or another algorithm with problem in hand. A possible and the most obvious way is to probe one algorithm and compare its performance to another one during problem solving process. In this case simple heuristics and meta-heuristics are out of competition, since once you solved the Optimization Problem you probably wouldn't optimize a second time. Here come **Hyper-Heuristics** to intelligently pick heuristics that is suitable to problem in hand. We will proceed with their description and how they deal with the NFLT consequences in following section.

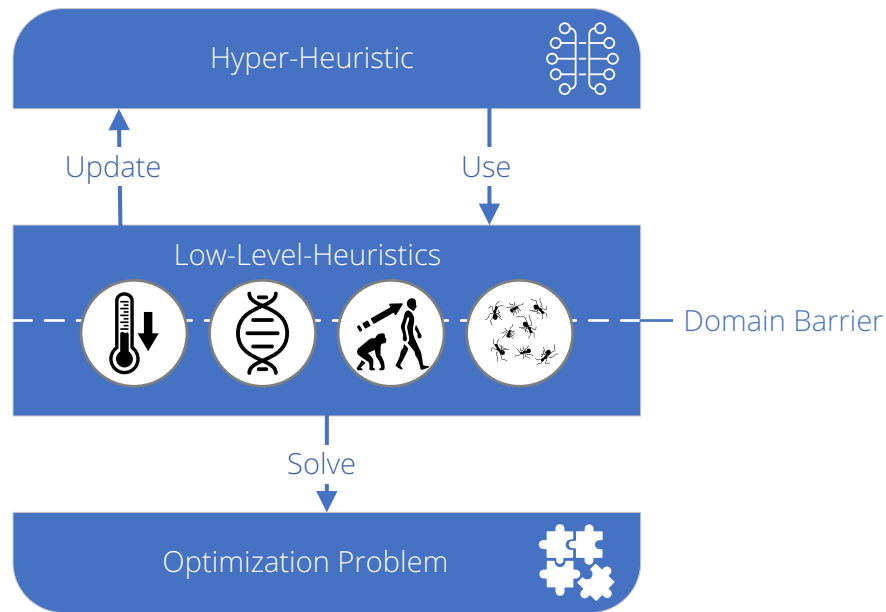


Figure 2.5 Hyper-Heuristics.

2.2.5 Hyper-Heuristics

A lot of state-of-the-art heuristics and meta-heuristics are developed in a complex and very domain-dependent way, which causes problems in an algorithm implementation reuse. It motivated research community to raise the level of generality at which the optimization systems can operate and still provide good quality Solutions for various Optimization Problems.

The term **Hyper-Heuristic** (HH) was defined to describe an approach of using some *High-Level-Heuristics* (HLH) to select over other *Low-Level-Heuristics* (LLH) and apply them to solve the *class of Optimization Problems* rather than particular instance. Indeed, scientists report that the combination of different HLH produces better results than if they were applied separately [24] — note previously discussed hybrid-heuristics. This behavior can be explained by the nature of search process and how it evolves in time. When you apply a heuristic, it sooner or later converge to some extreme point, hopefully global optimum. But it is 'blind' to other, not visited regions in the Search Space. Changing the trajectory of investigation by (1) drastically varying the Neighborhood, (2) changing the strategy of Neighborhood exploration and exploitation could (1) bring you to those previously unreachable zones (2) in more rapid ways. However, usually it is hard to predict how one LLH will behave in every stage of the search process in comparison to another. In Hyper-Heuristics, this job was encapsulated into the HLH and performed automatically.

In [64] authors made infer that Hyper-Heuristics can be viewed as a form of Reinforcement Learning, which is a logical conclusion especially if we rephrase it to *Hyper-Heuristics utilize Reinforcement Learning methodologies*.

The new concept which implicitly was used in Meta-Heuristics, but explicitly pointed out in Hyper-Heuristics is the **Domain Barrier**, see picture 2.5. As we told previously, HH do not tackle an OP directly, but use LLH instead. This means, that usually HH are minimally aware of the domain details, such as what are data types, relationships, etc. within a domain. This information rather encapsulated in LLHs and those HHs could be used to broader range of Optimization Problems.

¹Icons from thenounproject.com

With this idea, many researchers started to create not only Hyper-Heuristics to tackle a concrete optimization problem class, but also frameworks with building blocks for their creation.

Classification

Although, the research in Hyper-Heuristics field is actively ongoing, many algorithm instances were already created and some trials to organize approaches were conducted in [16, 24, 73].

Researchers in their surveys classify HHs by different characteristics, some of which overlap, but it also happens that important (from our perspective) features were not highlighted in all works.

In this section we present the union of those important Hyper-Heuristics classification facets to better justify the goal of this thesis.

We begin with the two broadest classes, which differentiate HH **routine**, also called as **nature of High-Level-Heuristic Search Space** [16, 17, 24]. The first class are Hyper-Heuristics to *select* Low-Level-Heuristic, in other words *Selection Hyper-Heuristic*. All our previous references to Hyper-Heuristics were made this concrete type. These algorithms operate in the Search Space, defined by complete and rather simple Low-Level-Heuristics that solve Optimization Problem. The task of HLH here is to pick the best suited LLH (or sequence of LLHs) based on available prior knowledge and apply it to OP underway. Note, that staging Hybrid-Heuristics could be viewed as Solutions of Selection HHs. Hyper-Heuristics of the second class seek to *construct* LLH following some predefined receipt and using the atomic components of other heuristics as Lego bricks. The other commonly used name here is *Construction Hyper-Heuristics*. These approaches often lead to creation of new and unforeseen heuristics that are expected to reveal good performance while solving the problem in hand.

Next, the distinction in **nature of Low-Level-Heuristics Search Space** arises. In other words: "How does the LLH derive Solutions for the OP?" Authors in [16, 17, 24] distinguished *construction* LLHs where Solution creation happens each time from scratch and *perturbation* LLHs where new Solutions created from parts of already existing ones.

The other broadly used characteristic is the **memory usage method**. From this perspective we distinguish Hyper-Heuristics in which the learning happens *on-line*, *off-line* or learning mechanisms are *not present* at all [16, 73].

- In **on-line** case, the HH derives an information, used to select among LLH, while those LLH are solving the problem.
- In **off-line** case, the learning happens before solving concrete Optimization Problem. Here one should first train an HH solving other homogeneous problem instances by underlying LLHs (off-line learning phase). After that, the HLH will be able to choose among LLHs and those be applicable to problem in hand (on-line use phase). Note, that this approach also requires creation of meta-features extraction mechanism and its application to every Optimization Problem.
- There exist also **mixed** cases, where learning happens first in off-line and later also in on-line phase. Definitely it is a promising (in terms of results quality) research direction, despite it high complexity.
- In the last case **no learning** mechanisms present, therefore HLH here performs some sort of Random Search over LLH Search Space. At first sight, it may look like

weak approach, but looking onto Variable Neighborhood Search Meta-Heuristic we would doubt it.

For more detailed analysis, description, other classification facets and respective Hyper-Heuristic examples we encourage reader to look into recent classification and surveying researches [15, 16, 24, 73].

Hyper-Heuristics Instance Examples

Hyper-Heuristic for Integration and Test Order Problem [38]. *HITO* is an example of generational HH. LLHs in this case are presented as a composition of basic EAs operators — crossover and mutation forming multi objective evolutionary algorithms (MOEA). HH selects those components from *jMetal* framework [25] using interchangeably Choice Function (in form of weighted linear equation) and Multi Armed Bandit based Heuristic to balance exploitation of good components and exploration of new promising ones.

Markov Chain Hyper-Heuristic [62]. *MCHH* is an on-line selective Hyper-Heuristic for multi-objective continuous problems. It utilizes reinforcement learning techniques and Markov Chain approximations to provide adaptive heuristic selection method. While solving an OP, *MCHH* updates prior knowledge about the probability of producing Pareto dominating Solutions by each underlying LLH using Markov Chains those guiding an LLH selection process. Applying on-line reinforcement learning techniques, this HH adapts transition of weights in the Markov Chains constructed from all available LLHs those updating prior knowledge for LLH selection.

Hyper-Heuristics Frameworks Examples

Hyper-Heuristics Flexible Framework [65]. *HyFlex* is a software skeleton, built specifically to help other researchers creating Hyper-Heuristics. It provides the implementation of components for 6 problem domains (Boolean Satisfiability, Bin Packing, Personnel Scheduling, Permutation Flow Shop, Traveling Salesman and Vehicle Routing problems), such as problem and solution descriptions, evaluation functions and adaptations for set of Low-Level-Heuristics. The set benchmarks and comparison techniques to other built HHs on top of *HyFlex* are included to framework as well.

The intent of *HyFlex* creators was to provide Low-Level features that enable others to focus directly on High-Level-Heuristics implementation without need to challenge other minor needs. It also brings a clear comparison among created HLH performance, since the other parts are mostly common. From the classification perspective, all derivatives from the *HyFlex* framework are Selection Hyper-Heuristics, however they utilize different learning approaches. Algorithms, built on top of *HyFlex* framework could be found in many reviews [24, 63, 73] or on the CHESC 2011 challenge website¹ (dedicated to choosing the best HH built on top of *HyFlex*).

Along with *HyFlex*, a number of hyper-heuristic-dedicated frameworks is growing, some of them are under active development while others are abandoned:

- **Hyperion [78]** is a construction hyper-heuristic framework aiming to extract information from OP search domain for identification of promising components in form of object-oriented analysis.

¹Cross Domain Heuristic Search Challenge website: asap.cs.nott.ac.uk/external/chesc2011/

- **hMod** [84] framework allows not only to rapidly prototype an algorithm using provided components, but also to construct those components using predefined abstractions (such as *IterativeHeuristic*). In current development stage, developers of *hMod* are focusing on creation of development mechanisms rather than providing a set pre-built heuristics.
- **EvoHyp** [70] framework focuses on hyper-heuristics created from evolutionary algorithms and their components. Here authors enable framework users to construct both selection and generation HHs for both construction and perturbation LLHs types.

2.2.6 Conclusion on Heuristic Solvers

To conclude our review on Heuristic approaches for solving Optimization Problems, we shortly remind you pros and cons of each heuristic level.

On the basis remain Simple Heuristics with all their domain-specific knowledge usage and particular tricks for solving problems. Usually, they are created to tackle a concrete problem instance in hand applying simple algorithmic approach. The simplicity of application and usually fast runtime paid back by medium results quality.

On the next level inhabit Meta-Heuristics. They could be compared with more sophisticated Solutions hunters which could not only charge directly, but also take a step back when stuck in a dead end. This additional skill enables them to survive in new and complex environments (Optimization Problems), however some adaptations to understand a concrete problem and parameter tuning for better performance still should be performed.

Among with MHs exit Hybrid-Heuristics. There is nothing special here, they just took some survival abilities from several Meta-Heuristics hoping to outperform and still requiring adaptation and tuning. In some cases this hybridization provides it advantage, but as the time shows, they did not force out MHs. Those we can conclude that the provided balance between development effort and exposed results quality not always assure users to use them.

Finally, the chosen ones that lead the others, Hyper-Heuristics are on the upper generality level. Operating by the other Heuristics, HHs analyze how good former are and make use of this knowledge by solving the concrete problem using those best suited Heuristics. Imposing such great abilities, Hyper-Heuristics tackle not only the concrete optimization problem, but entire class of problems.

2.3 Setting Heuristic Parameters

The most of existing learning algorithms expose some parameter set, needed to be assigned before using this algorithm. Modifying these parameters, one could change the system behavior and possible result quality.

When we are talking about the problem of settings the best parameters, following terms should be refined explicitly:

1. **Target System (TS)** is the subject which parameters are undergoing changes. Simply, it could be a heuristic, machine learning algorithm or other system.
2. **Parameter** is one of exposed by TS setting hooks. It should be described in terms of its type and possible values.

3. **Configuration** is the unique combination of parameter values, sufficient to run TS.
4. **Search Space** is the set of all possible Configurations for defined Parameters.

In this thesis we use notions of *Parameter* and *Hyper-Parameter* (HP) interchangeably, since the approaches discussed in this section are generally applicable also in Machine Learning cases. For instance, consider Neuron Network (NN). Hyper-parameters in this case will specify the structure (number of hidden layers, units, etc.) and learning process (learning rate, regularization parameters values, etc.) of network and changing them dramatically affect performance and results.

One frequent incumbent of Optimization Problems is **Parameter Settings Problem** (PSP) — searching of hyper-parameter values that optimize some characteristic of TS. When talking about NN example, PSP could be defined as task of maximization network accuracy in given dataset (Single Objective PSP). Taking into account a number of TS characteristics simultaneously, such as training time and prediction accuracy PSP transforms into Multi-Objective PSP.

The same applies to heuristics: proper assignment of hyper-parameters has a great impact on exploration-exploitation balance and those on algorithm performance.

Up until now there were formalized many approaches for solving task of settings hyper-parameters. One of the simplest and error-prone ways is just trusting your (or someones else) intuition and using those parameters that seems more or less logical for particular system under the problem instance. People quickly abandoned it in favor of automatic approaches, fortunately novel computational capacities easily provide a possibility for it. These automatic methods later could be split onto *off-line or parameter tuning* and *on-line or parameter control* techniques.

2.3.1 Parameter Tuning

Roughly speaking, the off-line approach is a process of traversing the search space of hyper-parameters and evaluating TS with these parameters on some set of toy problems. After finishing this process, the best found HPs are later used to solve new, unforeseen problem instance.

In this part of thesis we briefly outline existing automated approaches for parameter tuning illustrating them on a picture 2.6¹. In this example, the TS exposes two parameters: X_1 and X_2 . Each graphic shows dependencies between X_1 (horizontal axis) and X_2 (vertical axis) values and the subject of optimization along those axes (here depicted the maximization case). The best found configuration by each approach is highlighted in pink.

Grid Search Parameter Tuning. It is a rather simple approach for searching parameters. Here the original search problem is relaxed and later solved by brute-force algorithm. The set of all possible configurations (parameter sets) for relaxed problem is derived by specifying a finite number of possible values for each hyper-parameter under consideration. After evaluating all configurations on TS, the best found solution is reported. As you can see, this approach could skip promising parts of search space as shown on picture 2.6a.

Random Search Parameter Tuning. This methodology relies on random (often uniform) sampling of hyper-parameters and their evaluation on each iteration. At first

¹Original graphics are taken from [52]

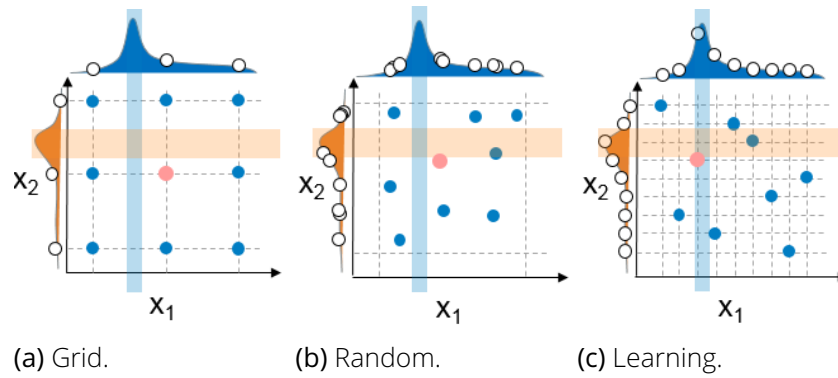


Figure 2.6 Automated Parameter Tuning Approaches.

sight, it might look unreliable to chaotically traverse the search space. But empirical studies show that with growing number of evaluations this technique starts to outperform grid search [4]: compare the best configurations (highlighted in pink) found by grid (picture 2.6a) and random search (picture 2.6b) techniques.

Model Based Search Parameter Tuning. In the most cases, the dependencies between tuned parameter values and optimization objective do exist and can be utilized for hyper-parameter tuning. By predicting which parameter combinations lead to better results, model-based tuning could make precise guesses. As it showed on picture 2.6c, at the beginning, this approach behaves as random search, but after accumulating enough information, learning algorithm starts making more precise guesses, which in contrast to previously described model-free approaches is desirable and more robust.

Naturally, this optimization problem could be tackled by almost every discussed here approach, however taking into account the facts that (1) TS here in most cases is a *black-box* we eliminate exact and approximate solvers, (2) the evaluation cost is huge, thus it is not desirable to apply any of described above heuristics directly.

With this idea in mind, researchers started to (1) create Bayesian optimization algorithms that traverse the search space more efficiently and (2) build models that could recreate the dependencies between parameters and objective values, a so-called *surrogate models*. While the former direction is nothing else but an enhancement to already existing optimization techniques, the later one is crucial for problems where objective is expensive to evaluate. The later is some sort of enhancement used in combination with former enabling you to simulate evaluation of real system instead of expensive direct evaluations. Still, it is a common approach to combine previously reviewed search space traversal techniques, such as Evolutionary Algorithms, Simulated Annealing, Tabu Search with surrogate models for optimization.

2.3.2 Systems for Model-Based Parameter Tuning

The parameter tuning is an obligatory task when the maximum system performance is a mast-have requirement and should be performed at the design time. Novel tuning approaches are usually built in form of frameworks with exposed hooks for attaching the system under estimation.

Since, the Target System evaluations here supposed to be extremely costly parameter tuning algorithms in those frameworks are trying to utilize every single bit of information from evaluations those building surrogate models and using Bayesian optimization approaches is obligatory.

In this section we review some among existing open-source parameter tuning systems from following perspectives:

- **Conditional parameters support** is (1) a provided for user and (2) handled by tuning system ability to describe conditional dependencies between hyper-parameters. As an example, imagine *crossover type* parameter of Genetic Algorithm that can take only some specific values: *Partially Mapped Crossover (PMX)*, *Cycle Crossover (CX)*, etc. Binding concrete crossover type, one will be required provide parameters for this crossover type, but eliminate respective parameters for other crossover types. This type of dependency could be described in form of parent-child relationship, however other types of dependencies also exists.
- **Parameter types support** is one of the basic usable tuning systems requirements. More concretely, Target System parameters could be not only numerical (integer or fractional), but also categorical in form of strings, boolean values, etc. Considering categorical data types, they could be either *nominal* (depict only possible atomic values) or *ordinal* (implies also value comparison, but no distance notion). For instance lets again analyze parameters of Genetic Algorithm: population size — numerical integer in range $[1 \dots \infty]$, mutation probability — numerical fractional $[0 \dots 1]$, crossover type — categorical nominal *PMX*, *CX*. Indeed, we could treat population size as a set of finite values 10, 100, 1000 — categorical ordinal parameter type.
- **Extensibility** is crucial when someone would like to try a new promising and intriguing learning algorithm for guiding a search, that was not available in parameter tuning system yet. Practically, one may need not only new learning algorithm, but some other features like non-trivial stopping criterion, tools for handling stochastic behaviors, or different strategies for random sampling (which are utilized while tuning system is learning before making a prediction).
- **Parallel Evaluations** required for utilizing available computing resources that could scale horizontally, those providing simultaneous evaluation of multiple Configurations which speeds-up the learning process.

Among reviewed systems we could distinguish ones that were created directly to face the parameter tuning problem and the others that are more generic optimizers but still applicable in parameter tuning cases. A concrete optimizer will be usable for searching the parameters, if it exposes several features. First, it must consider optimization function evaluation to be expensive and tackle this problem explicitly. For instance, using surrogate models or other Target System (TS) approximations. Next, potential tuner should be able to tackle dependencies and conditions among parameters.

SMACv3 [42]

Sequential Model-based Algorithm Configuration is a system for parameters tuning, developed by the AutoML research group (here we review *SMACv3*).

In their researches, scientists generalized the process of parameter tuning under the term *Sequential Model-Based Optimization* as iterations between steps of (1) fitting models and (2) using them to choice next Configurations for evaluation. We found that this term greatly formalizes all existing (to the best of our knowledge) parameter tuning approaches and those should be used as a distinguishing characteristic of tuning algorithms (since they naturally could be applied not only to parameter tuning problems).

SMAC is an extension introducing learning models and sampling mechanisms based on them to previously existing Random Online Aggressive Racing (*ROAR*) mechanism. Authors showed that the machine learning mechanisms and regression models in particular are applicable not only for tuning parameters, but also for optimizing any expensive black-box functions in general (playing the role of surrogate model).

The development of this system was motivated to tackle existing limitations of all published SMBO approaches, namely expanding an applicability not only to numerical, but also to categorical parameters and optimizing target algorithm not only on single, but on number of problem instances (benchmark set of problem instances) to reduce the influence of the variance.

A routine in *SMAC* could be imagined as iterated application two tree steps: (1) building learning model, (2) using it for making choices which Configurations to investigate next and (3) actual evaluation of sampled Configurations.

The evaluation (3) here carried out by original *ROAR* mechanism in which evaluation of each new candidate Solution continues until enough data (from benchmark set of problem instances) obtained to either replace current Solution or reject candidate. In contrary to original, model-less *ROAR*, *SMAC* at step (1) builds regression random forest — an instance of machine learning algorithm [14]. The usage of regression decision trees (which form the forest) motivated by known fact that they fit well to categorical data and complex dependencies in general. Later, at step (2) the Iterative Local Search (ILS) heuristic applies in combination with *Expected Improvement* (EI) evaluation (form of Bayesian optimization) [75]. ILS starts on the best previously found Configuration and traverses its Neighborhood distinguishing between neighboring Configurations using EI and regression model built at step (1). EI is large for those Configurations, which has low predicted cost and for those, with high uncertainty in results those naturally providing exploration-exploitation balance [46].

Exposing such a great learning capabilities and using Expected Improvement technique that guarantees converging to global optimum given enough time, the major drawback in this system is lack of flexibility to include conditional dependencies between parameters into sampling process.

In fact, used here search space representation framework *ConfigSpace*[57] is able to specify dependencies among hyper-parameters. However, to the best of our knowledge, during the Configuration sampling those conditions are not taken into account and could be broken thus resulting in illegal parameter combination. Those cases are handled and broken Configurations are rejected by *ConfigSpace*, but we believe that in case of ‘sparse’ search spaces (where significant amount of parameter combinations are restricted by conditions) this approach could lead to ineffective sampling and great struggling in system predictive abilities. Unfortunately, we did not find any officially published empirical studies of such cases and can only make guesses based on own intuition. However, the *SMACv3* developers advises for operation in such cases ¹ could serve as an evidence to correctness of our assumptions. One of possible solutions here could be the use of conditional-aware neighborhood definition in ILS (currently sampling is performed in regular Gaussian distribution).

ROAR mechanism is the derivative from *FocusedILS* algorithm (solver in *ParamILS* parameter tuning framework [43]) where each evaluation of new candidate Solution on problem instance performed sequentially. Since the *ROAR* evaluation strategy is applied at step (3), we conclude that the utilization of available computation capabilities is, possibly, another drawback of *SMACv3* framework.

¹Visit GitHub repository of *SMACv3* for more info <https://github.com/automl/SMAC3/issues/403>

IRACE [58]

The first system under analysis is an implementation of Iterated Racing Algorithm [11] in *IRACE* hyper-parameter tuning package.

The underlying methodology is somehow similar to one implemented in *SMACv3* and consists of three main steps: (1) sampling new Configurations using prior knowledge, (2) finding the best ones among sampled using racing algorithm and (3) updating the prior knowledge to bias future samples in (1) towards better Configurations. Prior knowledge here represented as probability distributions for each parameter independently (truncated normal and discrete distributions for numeric and categorical hyper-parameters respectively).

Iterated racing step (2) here is a process of running Target System using sampled Configuration on a set of problem instances. After solving each instance, the statistically worse-performing Configurations are rejected and racing proceeds with remaining ones. This process continues until reaching the required number of survivals or after solving a requiring amount of problem instances (in this case all remaining Configurations are considered to be good).

IRACE supports various data types, such as numerical or categorical and the possibility of conditions description as well. While the problem of data types solved by different underlying distributions, the conditional relationships are handled by the dependency graphs. During sampling (1), first are sampled non-conditional and only afterwards, if respective conditions are satisfied, sampled dependent parameters.

The framework highly relies on racing algorithm for parallel evaluations and on *Friedman test* [20] or alternatively *paired t-test* for statistical analysis of Configurations, those it is static in terms of variability and extensibility of learning mechanisms. In terms of parallel evaluations, the algorithm utilizes all available resources at the beginning of each racing step, however as the process continues, fewer evaluations are executed in parallel those available resources are idling and not utilized optimally at all steps of algorithm.

BOHB [29]

While *SMAC* outperforms and partially reuses decisions done in *ParamILS*, *BOHB* (Bayesian Optimization combined with HyperBand) is the parameter tuning tool that outperforms *SMAC* and was suggested by the same AutoML research group.

As it stated in name, the *SMBO* routines in this framework are carried out with mainly two algorithms: learning (1) and Configurations sampling (2) done by Tree Parzen Estimator (TPE) — Bayesian Optimization technique, while evaluation of sampled Configurations and their comparison are carried out by HyperBand (HB) algorithm.

The TPE instead of naïve Gaussian Processes-based (GP BO) Bayesian Optimization and Expected Improvement evaluation, was motivated by better dimensional scaling abilities and internal support of both numerical and categorical data types (however, some minor transformations are still required). Unlike vanilla BO, where the optimization done by modeling results distributions given Configuration parameters, TPE builds two distributions of parameters splitting Configurations into two sets according to their ‘goodness’ and later proposing those parameters with high probability to be in ‘good’ distribution and simultaneously low probability to be in ‘bad’ distribution. For more detailed explanation we refer to TPE description, given in [5].

The central part of *BOHB*, namely HyperBand is a promising multi-armed bandit strategy for hyper-parameter optimization [56] in which the *budget* for Configurations evaluation is defined beforehand and divided into iterations. The role of budget could play any control parameter that denotes the accuracy of Configuration evaluation

by TS, where estimation with the maximum budget gives you the most precise Configuration evaluation while minimum amount of budget results in the least accurate approximation of Configuration evaluation result. Running examples of budget could be the number of iterations in iterative algorithm, epochs to train the neuron network or number of problem instances from benchmarking set.

As the result, requirements arise for TS to expose and support budget usage as expected in *BOHB*.

At each iteration, HB randomly samples a number of Configurations which, in fact, decreases for later iterations while the amount of budget remains the same. As the outcome, first iterations of HB are full of coarse-grain evaluated Configurations, while later iterations produce higher number of more accurate measurements. Each iteration of HB is split to number of Successful Halving (SH) procedure executions which drop (usually $\frac{2}{3}$) badly performing Configurations. As one could expect, since the number of measuring Configurations in each iteration decreases, the amount of SHs execution drops too and those the remaining Configurations are evaluated more precisely.

The binding of HyperBand and Bayesian Tree Parzen Estimator made is several places. Firstly, the learning models are updated each time when new results are available for every budget value. Next, at each iteration of HB, TPE model is used for sampling new Configurations. Note that *BOHB* uses models, built on results obtained with the largest budget only. This decision leads in more precise predictions in the later stages of parameter tuning process.

The drawback of this system lays in the way of handling conditions between hyper-parameters. Distributed HyperBand implementation on Steroids (HpBandSter) ¹ is a *BOHB* particular implementation uses *ConfigSpace* framework (as *SMACv3*) for Configuration Space definition. As we discussed in *SMACv3* description (section 2.3.2), *ConfigSpace* naturally allows to encode the dependencies and conditions among parameters. As authors stated, the TPE learning models are also able to learn these dependencies implicitly by shrinking the densities for forbidden parameters (actually those parameter values are still added to models by *imputation* mechanism where default values are used for forbidden parameters). However, consider a case of two Configurations C_1 , C_2 appearance such that some parameter P_i is forbidden in C_1 , but not in C_2 , which will break the densities and as a consequence, the prediction performed using such distributions will often result in Configurations with violated parameter dependencies within 'sparse' search spaces, hurting the performance and accuracy of sampling. The actual number of such cases could vary dramatically in 'sparse' Search Spaces those distributions estimated by KDEs will not reflect the reality.

BRISv2 [71]

The great part of software potential lays not only in its ability to tackle a problem in hand, but also on the general usability and adaptivity to unforeseen tasks. Here we review BRISv2 of version 2¹, since the very early BRISv2 versions (major 1 [72]) were more monolithic and those hard to apply for parameter tuning problem in hand.

While designing this system, authors were focused on learning mechanisms for parameter prediction and overall system modularity as well. Being a Software Product Line, BRISv2 was designed as a set of interacting components (nodes) each acting according to its own specific role. The system could be viewed from two perspectives.

¹GitHub repository: <https://github.com/automl/HpBandSter>

¹GitHub repository: <https://github.com/dpukhkaiev/BRISv2/releases/tag/v2.3.0>

One is general outline of all available nodes and the other is analysis of *main-node* concretely.

Before reviewing each part, it is worth to justify formalized in system terms. Note, that they are pretty the same as we defined above, but here most of them also have been implemented in form of classes.

1. *Experiment* denotes a complete run of BRISE, for instance parameter tuning session. Handles such information as BRISE Experiment Description (JSON-like specification of parameter tuning procedure), all evaluated during session configurations and derived from description search space.
2. *Configuration* is a combination of input Parameter values for target system (or algorithm) under evaluation. Could be evaluated several times (to obtain statistical data) those contain number of Tasks and compared against other Configurations.
3. *Parameter* is a meta-description of concrete Configuration part and a building block for Search Space, defines a set or range of possible values.
4. *Search Space* comprises all Parameters and their dependencies, could verify the validity of Configuration.
5. *Task* is an evaluation of TS under provided Configuration for statically specified in Description scenario.

From a birds eye view perspective, BRISE consists of *main-node* as the system backbone, several *worker-nodes* as target algorithm runners under provided Configurations, *worker-service* to distribute tasks load between set of worker-nodes, *front-end-node* to control and report optimization process on a web-page, and non-obligatory *benchmark-node* that could be handy for executing and analyzing number of Experiments.

To use BRISE for parameter tuning, one should (1) construct an Experiment and Search Space json-like descriptions and (2) add the respective function for target system evaluation in *workers* with provided Task. All the rest will be carried out by system.

From our perspective, the advantage of BRISE over other systems comes from the design of *main-node*. It is a set of operating core entities (Experiment, Search Space and Configuration) with other non-core entities exposed to user for variability, such as surrogate models, termination criteria, outliers detectors, repetition strategies, etc. Naturally, a number of implementations are provided out-of-box for all variable entities. For instance, two prediction models: ridge regression with polynomial features and TPE BO; several termination criteria: wall-clock timeout, exceeding number of evaluated Configurations, absence of improvement and others. Using different combinations entities, one could obtain the system with desired behavior or even further, supply own learning models, sampling plans, enhance termination logic, etc.

As every other system, BRISE also has flaws. Talking about the search space representation, BRISE relies on ConfigSpace module [57] which is also used SMACv3 and BOHB frameworks. The predictive models are acting pretty the same, so all previous claims are also valid here — it is hard to apply BRISE in cases of sparse Configurations spaces.

Generally, the biggest disadvantage of Parameter Tuning defined by the fact, that it usually requires large number of target algorithm runs to evaluate its performance with different settings. On opposite site, the parameter control approaches are solving this issue but the drawback lays in their universality.

2.3.3 Parameter Control

One advantageous characteristic of the system is its ability to adapt in runtime. It could happen, that tuned algorithm performs well in early beginning of new problem solving, but later could struggle hard while the other configuration may result in opposite behavior. It could be caused by various reasons, and it is usually hard to tell, which of them you are facing at concrete moment.

In contrast to parameter tuning, where optimal parameters first were searched and only later applied to problem in hand, the parameter control is an approach of searching the best system configuration, while solving the problem in hand. It also could be expressed as the ability of system to react on changes in the solving process, in other words it is known as *on-line* version of parameter tuning. The drawback of this approach lays in lack of generality, since often the parameter control technique is algorithm dependent.

Searching the examples of parameter control for heuristics, one definitely will find himself exhausted to enumerate all proposed methodologies for evolutionary algorithms, since the origins of this idea lay in EAs [48]. However, techniques for other meta-heuristics do exist too. As far as we know, the approach to adapt parameters mostly dependent onto concrete algorithm instance, those it is hard to present a general classification of approaches for all algorithms, but it could be done for each particular family.

The only one brute classification facet we able to distinguish is the type of control mechanism, where *deterministic* (following a predefined schedule) and *adaptive* (assign parameter values upon received feedback) types are distinguishable.

We provide an insight of parameter control by reviewing some examples among existing adaptation strategies in meta-heuristics. Generally speaking, authors reported significant improvements in optimization performance for all mentioned control mechanisms in every heuristic. For more comprehensive review of recently published parameter control strategies in meta-heuristics we encourage reader to examine [41].

Parameter Control in Evolutionary Algorithms

The motivation for vast amount of performed studies tackling adaptive changes in Evolutionary Algorithms is that the performance of later greatly depends not only on (1) parameter values, which were found by tuning approaches but also on (2) changing them during the run, to help the algorithm converge. Shortly, there exist strategies to control each EA parameter separately or in groups. The most commonly implicated parameters are: *population size*, *selection strategies* and *variation aspects* (namely crossover and mutation operators). For more detailed explanation and analysis of each approach we suggest reader to refer recently conducted reviews and researches dedicated to parameter control in Evolutionary Algorithms [1, 48, 76].

The only common distinction we found here is that listed previously mechanisms are extended by 3rd control type — *self-adaptive*, which implies encoding of mentioned parameter values in solution genomes those allowing them to co-evolve with solutions in the runtime.

Parameter Control in Simulated Annealing

In Simulated Annealing, however, the most frequently controlled parameters are the *cooling schedule* (the velocity of temperature decrease) and the *acceptance criteria* (decide whether to accept proposed solution, or not).

The cooling rate parameter control is motivated by the fact, that if the temperature decreasing too rapidly, the optimization process may stuck quickly in local optima, while too low cooling rate is computationally costly, since it requires more evaluations to converge. Many researches were made to investigate which is the best *deterministic* strategy among linear, exponential, proportional, logarithmic or geometrical. In contrast, authors in [47] proposed the adaptive strategy to change the cooling rate, based on statistical information evaluated on each optimization step. In details, if the statistical analysis (named in research a *heat capacity*) shows that system is unlikely to be trapped in local optima — cooling rate increased, but if the possibility of being trapped into local optima is high — the cooling rate decreased.

The other example shows the adaptation of acceptance criteria in [35]. It is based on thermodynamics fundamentals, such as *entropy* and *kinetic energy*. Authors suggest replacing the standard acceptance criteria (based on current temperature and degradation of solution quality) with the criteria based on change in solutions entropy.

For other approaches of parameter control in Simulated Annealing meta-heuristic we suggest to check [21, 44].

2.4 Combined Algorithm Selection and Hyper-Parameter Tuning Problem

As we can observe, the goals of automatic machine learning are quite similar to persecuted by Hyper-Heuristics.

They both operate on Search Space of algorithms (or their building blocks) which later are combined, with an objective to find the best performing one, and used to solve the problem in hand.

In this section we review one particular representative of automatic machine learning systems. Based on ML framework Scikit-learn [69], Auto-sklearn system [30] operates over number of classifiers, data and features preprocessing methods **including their hyper-parameters** to construct, for given dataset, the best performing (in terms of classification accuracy) machine learning pipeline. This problem was formalized as *Combined Algorithm Selection and Hyper-parameter tuning problem* (CASH) and presented previously in Auto-WEKA [80] system. Intuitively it could be rephrased as follows: “For given optimization problem, find the best performing algorithm and its hyper-parameters, among available, and solve the problem”.

Note, how Auto-sklearn is in some sort similar to Hyper-Heuristics, which use LLHs for traversing a Search Space and solving the OP. CASH problem seems to us as union of problems solved by Hyper-Heuristics and Parameter Tuning approaches. We also found that the Architecture Search problems [28] (related to Neural Networks) are nothing else, but particular case of CASH.

Turning back to Auto-sklearn, the crucial decisions made here is the combination of off-line and on-line learning, resulted into state-of-art performance of Auto-sklearn in classification tasks.

During the off-line phase, for each of available dataset, published by OpenML community [31], the search of best performing machine-learning pipeline was done using Bayesian Optimization technique implemented in discussed previously SMAC [42] framework. After that, the *meta-learning* executed to conduct the meta-features for each dataset. The entropy of results, data skewness, number of features and their classes is a sparse set of meta-features used to characterize a dataset (overall number is 38).

The resulting combination of dataset, machine learning pipeline and meta-features

were stored and later used to seed the on-line phase of pipeline search. The information from meta-learning phase is used as follows: for a given new dataset, system derives the meta-features and selects some portion of created during off-line phase pipelines, that are the nearest in terms of meta-feature space. Then these pipelines are evaluated on a new dataset to seed the Bayesian Optimization in SMAC, which results in ability to evaluate, in principle, well-performing Configurations at the beginning of tuning process.

During the on-line phase, the other crucial improvement was introduced. Usually, while searching the best-performing pipeline, a lot of effort spent to build, train and evaluate intermediate ones. After each evaluation, only the results and pipeline description are stored, but the pipeline itself is discarded. In Auto-sklearn, however, the idea lays in preserving previously instantiated and trained pipelines, obtained while solving the CASH problem. Later they are used to form an ensemble of models and tackle final problem together. This mean, that the results of this architecture search is a set of models with different hyper-parameters and preprocessing techniques, rather than one model. This ensemble starts from the worst performing ones (obtained at the search beginning) and ending by the best suited for dataset in hand. Naturally, their influence on final results are weighted.

However, the potential of off-line phase is derived entirely from the existence of such dataset repository and depends on availability of homogeneous datasets. The proposed on-line methodology, which mimics the regression trees, is more universal and could be reused widely.

In general, the empirical investigation of proposed approach universality would be rather intriguing, since the only cases of Auto-sklearn application we found are the classification tasks, but not a regression problems [8, 30].

The field of automated machine learning is one of trending research directions those there exist dozens open-source systems, such as *Auto-Weka* [80], *Hyperopt-Sklearn* [53], *Auto-Sklearn* [30], *TPOT* [66], *Auto-Keras* [45], etc. Among open-source, there are many commercial auto-ml systems, such as *RapidMiner.com*, *DataRobot.com*, Microsoft's *Azure Machine Learning*, Google's *Cloud AutoML*, and Amazon's *Machine Learning on AWS*.

2.5 Conclusion

We believe, that a central role of off-line selective Hyper-Heuristics, namely solving the algorithm selection problem could be represented as the parameter tuning problem, where instead of parameter selection we select entire system according to reported performance. With this idea in mind, we investigate a possibility of turning parameter tuning system into a Hyper-Heuristic. For doing so, our next step is the review and comparison of existing parameter tuning approaches and systems.

It could be compared with a warp-engine usage on a spacecraft, which gives the possibility to travel faster than speed of light by orders of magnitude. It is definitely fast enough, however if you consider a space-folding engine for instant traveling and use it to jump onto a huge distance, while warp-engine is

The meta-heuristic systems designers reported positive impact of parameter control embedding. However, as the outcome of the no-free-lunch theorem, those systems can not tolerate broad range of problems, for instance, problem classes. In other hand, hyper-heuristics are designed with an aim to select the low level heuristics and those propose a possible solution of problem, stated in no-free-lunch theorem, but the lack of parameter control could dramatically decrease the performance of LLH (probably, I need to find a prove of this, or rephrase).

Scope of thesis defined. In this thesis we try to achieve the best of both worlds applying the best fitting LLH and tuning it's parameters while solving the problem on-line.

3 Concept Description

In this chapter we describe the concept of developed selection Hyper-Heuristic with parameter control, not diving deep into the implementation details.

The structure of this chapter is as follows.

Yevhenii: maybe we should not highlight the structure of such a small chapter

First, in in section 3.1 we define the Search Space entity requirements and structure. It should bound the world of Low Level Heuristics and the world of Hyper-parameters of those heuristics.

Next, we describe the Prediction process within the previously defined Search Space in section 3.2. Here we highlight an importance of a prediction model decoupling from the previously defined Search Space structure. Doing so, we provide certain level of flexibility for user in the usage of different prediction models or developing his own.

Finally, in section 3.3 we gather our attention onto the Low Level Heuristics - a working horse of the hyper-heuristic. Here we highlight the requirements for LLH in terms of features that will be used by HH.

3.1 Search Space

Importance explanation

Required structure feature-tree structured

3.2 Prediction Process

Importance explanation

Requirements generality, top-down approach of optimization – different views of same Configuration (level-dependent) - filtering, transformation – consider problem features? while selecting meta-heuristic [50] page 6

3.3 Low Level Heuristics

Importance explanation

Requirements

3.4 Conclusion of concept

to be done...

4 Implementation Details

In this chapter we dive into the implementation details of the selection hyper-heuristic with parameter control.

The best practice in software engineering is to minimize an effort for the implementation and reuse already existing and well-tested code. With this idea in mind we had decided to reuse one of existing (and highlighted by us in ??) open-source hyper-parameter tuning systems as the code basis and those turn it into the core of hyper-heuristic. Do to so we analyze the existing systems and highlight important non-functional characteristics from the implementation perspective in section 4.1. Since the selected code base system is not the ideal in terms of such features as Search Space entity abilities and the prediction process, we consider some adaptations in sections 4.2 and 4.3 respectively. We also reuse the set of Low Level Heuristics in section 4.5.2.

4.1 Hyper-Heuristics Code Base Selection

A.k.a. "brain". Need to find a better way to call this part of HH..

4.1.1 Requirements

4.1.2 Parameter Tuning Frameworks

SMAC

BOHB

IRACE

BRISv2

Yevhenii: Maybe, smth else..

4.1.3 Conclusion

BRISv2 is the best system for code basis, however it has to be changed as we describe in following sections.

4.2 Search Space

4.2.1 Base Version Description

What is the problem with the current Search Space?

The Scope Refinement Work Throw away and write a new one :D

4.2.2 Implementation

Description

Motivation of structure

Class diagram - i think, I will put it into the appendix

4.3 Prediction Logic

4.3.1 Base Version Description

The Scope Refinement Work prediction should be done in feature-tree structured search space. Most models could handle only flat search space and we would like to enable reuse of those existing models. Though we decouple the structure of Search Space in entity **Predictor**, while actual prediction process is done in underlying models, that Predictor uses.

4.3.2 Predictor

to decouple prediction from structure of search space.

4.3.3 Prediction Models

Tree parzen estimator

Multi Armed Bandit

Sklearn linear regression wrapper

4.4 Data Preprocessing

4.4.1 Heterogeneous Data

description and motivation of data preprocessing notions

4.4.2 Base Version Description

and Scope of work analysis

4.4.3 Wrapper for Scikit-learn Preprocessors

4.5 Low Level Heuristics

4.5.1 Requirements

4.5.2 Code Base Selection

Available Meta-heuristics with description of their current state With the aim of effort reuse, the code base should be selected for implementation of the designed hyper-heuristic approach.

SOLID

MLRose

OR-tools

pyTSP

LocalSolver

jMetalPy

4.5.3 Scope of work analysis

opened PR

4.6 Conclusion

5 Evaluation

5.1 Evaluation Plan

5.1.1 Optimization Problems Definition

Traveling Salesman Problem

tsplib95 benchmark set which problem I want to solve with hyper-heuristic

N-Queens Problem?

Knapsack Problem?

5.1.2 Hyper-Heuristic Settings

To evaluate the performance of developed system we first need to compare it with the base line. In our case it is the simple meta-heuristic that is solving the problem with static hyper-parameters.

In order to organize the evaluation plan, we distinguish two stages of setup, where different approaches could be applied. At the first stage we select Low Level Heuristic, while at the second one we select hyper-parameters for LLH. The approaches for each step are represented in table 5.1.

Table 5.1 System settings for benchmark

Low Level Heuristics selection	LLH Hyper-parameters selection
1. Random	1. Default
2. Multi Armed Bandit	2. Tuned beforehand
3. Sklearn Bayesian Optimization	3. Random
4. Static selection of SA, GA, ES	4. Tree Parzen Estimator
	5. Sklearn Bayesian Optimization

For instance, mentioned above baseline could be described as *Settings*4.1. for meta-heuristics with default hyper-parameters and as *Settings*4.2. for meta-heuristics with tuned beforehand hyper-parameters.

For our benchmark we selected following settings sets:

- *Baseline*: 4.1, 4.2;

- *Random Hyper-heuristic*: 1.1, 1.2, 1.3, 4.3;
- *Parameter control*: 4.4, 4.5;
- *Selection Hyper-Heuristic*: 2.1, 3.1, 2.2, 3.2;
- *Selection Hyper-Heuristic with Parameter Control*: 2.4, 2.5, 3.4, 3.5;

Each of these settings will be discussed in detail in the following section.

5.1.3 Selected for Evaluation Hyper-Heuristic Settings

Baseline

Hyper-heuristic With Random Switching of Low Level Heuristics

Parameter control

Selection Only Hyper-Heuristic

Selection Hyper-Heuristic with Parameter Control

5.2 Results Discussion

5.2.1 Baseline Evaluation

Meta-Heuristics With Default Hyper-Parameters

Meta-Heuristics With Tuned Hyper-Parameters

Results Description and Explanation

5.2.2 Hyper-Heuristic With Random Switching of Low Level Heuristics

Results Description and Explanation

5.2.3 Parameter Control

Results Description and Explanation

5.2.4 Selection Only Hyper-Heuristic

auto-sklearn paper, p.2 - comparison of GP and TPE BOs.

Results Description and Explanation

5.2.5 Selection Hyper-Heuristic with Parameter Control

Results Description and Explanation

5.3 Conclusion

6 Conclusion

Reviewer: answer research questions

comparison to HITO [38]

Hyper-Heuristics with parameter tuning (or better say, control?), Constrained Parameter Tuning and Architecture search problems all are the same? All these problems are seems to talk about the same thing, and trying to solve it in the same ways, while calling it differently. In one hand, it could be the result of relatively young research direction (in all cases). In other hand we could make such an assumption because knowledge lack \smile .

Hyper-Heuristics and Automatic Machine Learning are the same.

7 Future work

add more sophisticated models

dependencies / constraints in search space

add new class of problem (jmetalpy easily allows it)

evaluation on different types and classes

adaptive time for tasks

bounding LLH by number of evaluations, not time

interesting direction: apply to automatic machine learning problems solving, compare to auto-sklearn.

technique to optimize obtained surrogate model should be generalized

investigate more deeply decoupling of data preprocessing and learning algorithm
auto-sklearn

influence for warm-start onto this kind of HH (by off-line learning) Although, the influence of meta-learning, applied in Auto-Sklearn system [30] to warm-start the learning mechanism proved to worth the effort spent, as well as it was reported by developers of Selective Hyper-Heuristics with mixed type of learning [83], it is intriguing to check an influence of metal-learning onto Selective Hyper-Heuristics with Parameter Control.

Random Forest HLH surrogate model

Reviewer: consider merging with conclusion, if too short

Bibliography

- [1] Aldeida Aleti and Irene Moser. "A systematic literature review of adaptive parameter control methods for evolutionary algorithms". In: *ACM Computing Surveys (CSUR)* 49.3 (2016), pp. 1–35.
- [2] Satyajith Amaran et al. "Simulation optimization: a review of algorithms and applications". In: *Annals of Operations Research* 240.1 (2016), pp. 351–380.
- [3] David L Applegate et al. *The traveling salesman problem: a computational study*. Princeton university press, 2006.
- [4] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *Journal of machine learning research* 13.Feb (2012), pp. 281–305.
- [5] James S Bergstra et al. "Algorithms for hyper-parameter optimization". In: *Advances in neural information processing systems*. 2011, pp. 2546–2554.
- [6] Hans-Georg Beyer and Hans-Paul Schwefel. "Evolution strategies—A comprehensive introduction". In: *Natural computing* 1.1 (2002), pp. 3–52.
- [7] Leonora Bianchi et al. "A survey on metaheuristics for stochastic combinatorial optimization". In: *Natural Computing* 8.2 (2009), pp. 239–287.
- [8] A. Biedenkapp et al. "Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework". In: *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*. June 2020.
- [9] Lorenz T Biegler and Ignacio E Grossmann. "Retrospective on optimization". In: *Computers & Chemical Engineering* 28.8 (2004), pp. 1169–1192.
- [10] Mauro Birattari et al. "Classification of Metaheuristics and Design of Experiments for the Analysis of Components Tech. Rep. AIDA-01-05". In: (2001).
- [11] Mauro Birattari et al. "F-Race and iterated F-Race: An overview". In: *Experimental methods for the analysis of optimization algorithms*. Springer, 2010, pp. 311–336.
- [12] Jacek Błażewicz, Wolfgang Domschke, and Erwin Pesch. "The job shop scheduling problem: Conventional and new solution techniques". In: *European journal of operational research* 93.1 (1996), pp. 1–33.
- [13] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. "A survey on optimization metaheuristics". In: *Information Sciences* 237 (2013), pp. 82–117.
- [14] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.
- [15] Edmund Burke et al. "Hyper-heuristics: An emerging direction in modern search technology". In: *Handbook of metaheuristics*. Springer, 2003, pp. 457–474.

- [16] Edmund K Burke et al. "A classification of hyper-heuristic approaches: revisited". In: *Handbook of Metaheuristics*. Springer, 2019, pp. 453–477.
- [17] Edmund K Burke et al. "Hyper-heuristics: A survey of the state of the art". In: *Journal of the Operational Research Society* 64.12 (2013), pp. 1695–1724.
- [18] Taesu Cheong and Chelsea C White. "Dynamic traveling salesman problem: Value of real-time traffic information". In: *IEEE Transactions on Intelligent Transportation Systems* 13.2 (2011), pp. 619–630.
- [19] Wikimedia Commons. *File:Metaheuristics classification fr.svg* — *Wikimedia Commons the free media repository*. 2017.
- [20] William Jay Conover and William Jay Conover. "Practical nonparametric statistics". In: (1980).
- [21] Juan De Vicente, Juan Lanchares, and Román Hermida. "Placement by thermodynamic simulated annealing". In: *Physics Letters A* 317.5-6 (2003), pp. 415–423.
- [22] Kalyanmoy Deb. "Multi-objective optimization". In: *Search methodologies*. Springer, 2014, pp. 403–449.
- [23] Marco Dorigo. "Ant colony optimization". In: *Scholarpedia* 2.3 (2007), p. 1461.
- [24] John H Drake et al. "Recent advances in selection hyper-heuristics". In: *European Journal of Operational Research* (2019).
- [25] Juan J Durillo and Antonio J Nebro. "jMetal: A Java framework for multi-objective optimization". In: *Advances in Engineering Software* 42.10 (2011), pp. 760–771.
- [26] AE Eiben and JE Smith. "Popular Evolutionary Algorithm Variants". In: *Introduction to Evolutionary Computing*. Springer, 2015, pp. 99–116.
- [27] Agoston E Eiben and James E Smith. "What is an evolutionary algorithm?" In: *Introduction to Evolutionary Computing*. Springer, 2015, pp. 25–48.
- [28] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural architecture search: A survey". In: *arXiv preprint arXiv:1808.05377* (2018).
- [29] Stefan Falkner, Aaron Klein, and Frank Hutter. "BOHB: Robust and efficient hyperparameter optimization at scale". In: *arXiv preprint arXiv:1807.01774* (2018).
- [30] Matthias Feurer et al. "Efficient and robust automated machine learning". In: *Advances in neural information processing systems*. 2015, pp. 2962–2970.
- [31] Matthias Feurer et al. "OpenML-Python: an extensible Python API for OpenML". In: *arXiv* 1911.02490 ().
- [32] Goncalo Figueira and Bernardo Almada-Lobo. "Hybrid simulation–optimization methods: A taxonomy and discussion". In: *Simulation Modelling Practice and Theory* 46 (Aug. 2014).
- [33] Fedor V Fomin and Petteri Kaski. "Exact exponential algorithms". In: *Communications of the ACM* 56.3 (2013), pp. 80–88.
- [34] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 174. freeman San Francisco, 1979.
- [35] Kambiz Shojaee Ghandeshtani and Habib Rajabi Mashhadi. "An entropy-based self-adaptive simulated annealing". In: *Engineering with Computers* (2019), pp. 1–27.
- [36] Fred Glover. "Tabu search—part I". In: *ORSA Journal on computing* 1.3 (1989), pp. 190–206.

- [37] Oded Goldreich. *P, NP, and NP-Completeness: The basics of computational complexity*. Cambridge University Press, 2010.
- [38] Giovanni Guizzo et al. "A hyper-heuristic for the multi-objective integration and test order problem". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 2015, pp. 1343–1350.
- [39] Pierre Hansen and Nenad Mladenović. "Variable neighborhood search". In: *Handbook of metaheuristics*. Springer, 2003, pp. 145–184.
- [40] Robert Hooke and Terry A Jeeves. "Direct Search" Solution of Numerical and Statistical Problems". In: *Journal of the ACM (JACM)* 8.2 (1961), pp. 212–229.
- [41] Changwu Huang, Yuanxiang Li, and Xin Yao. "A Survey of Automatic Parameter Tuning Methods for Metaheuristics". In: *IEEE Transactions on Evolutionary Computation* (2019).
- [42] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. "Sequential model-based optimization for general algorithm configuration". In: *International conference on learning and intelligent optimization*. Springer. 2011, pp. 507–523.
- [43] Frank Hutter et al. "ParamLLS: an automatic algorithm configuration framework". In: *Journal of Artificial Intelligence Research* 36 (2009), pp. 267–306.
- [44] Lester Ingber. "Adaptive simulated annealing (ASA): Lessons learned". In: *arXiv preprint cs/0001018* (2000).
- [45] Haifeng Jin, Qingquan Song, and Xia Hu. "Auto-Keras: An Efficient Neural Architecture Search System". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2019, pp. 1946–1956.
- [46] Donald R Jones, Matthias Schonlau, and William J Welch. "Efficient global optimization of expensive black-box functions". In: *Journal of Global optimization* 13.4 (1998), pp. 455–492.
- [47] Mariia Karabin and Steven J Stuart. "Simulated Annealing with Adaptive Cooling Rates". In: *arXiv preprint arXiv:2002.06124* (2020).
- [48] Giorgos Karafotias, Mark Hoogendoorn, and Ágoston E Eiben. "Parameter control in evolutionary algorithms: Trends and challenges". In: *IEEE Transactions on Evolutionary Computation* 19.2 (2014), pp. 167–187.
- [49] James Kennedy and Russell Eberhart. "Particle swarm optimization". In: *Proceedings of ICNN'95-International Conference on Neural Networks*. Vol. 4. IEEE. 1995, pp. 1942–1948.
- [50] Pascal Kerschke et al. "Automated algorithm selection: Survey and perspectives". In: *Evolutionary computation* 27.1 (2019), pp. 3–45.
- [51] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. "Optimization by simulated annealing". In: *science* 220.4598 (1983), pp. 671–680.
- [52] Patrick Koch et al. "Automated hyperparameter tuning for effective machine learning". In: *Proceedings of the SAS Global Forum 2017 Conference*. 2017.
- [53] Brent Komer, James Bergstra, and Chris Eliasmith. "Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn". In: *ICML workshop on AutoML*. Vol. 9. Citeseer. 2014.
- [54] John R Koza. "Evolution of subsumption using genetic programming". In: *Proceedings of the First European Conference on Artificial Life*. 1992, pp. 110–119.
- [55] Niklas Lavesson and Paul Davidsson. "Quantifying the impact of learning algorithm parameter tuning". In: *AAAI*. Vol. 6. 2006, pp. 395–400.

- [56] Lisha Li et al. "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6765–6816.
- [57] M. Lindauer et al. "BOAH: A Tool Suite for Multi-Fidelity Bayesian Optimization & Analysis of Hyperparameters". In: *arXiv:1908.06756 [cs.LG]* (2019).
- [58] Manuel López-Ibáñez et al. "The irace package: Iterated racing for automatic algorithm configuration". In: *Operations Research Perspectives* 3 (2016), pp. 43–58.
- [59] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. "Iterated local search". In: *Handbook of metaheuristics*. Springer, 2003, pp. 320–353.
- [60] Silvano Martello and Paolo Toth. "Bin-packing problem". In: *Knapsack problems: Algorithms and computer implementations* (1990), pp. 221–245.
- [61] Olivier C Martin and Steve W Otto. "Combining simulated annealing with local search heuristics". In: *Annals of Operations Research* 63.1 (1996), pp. 57–75.
- [62] Kent McClymont and Edward C Keedwell. "Markov chain hyper-heuristic (MCHH) an online selective hyper-heuristic for multi-objective continuous problems". In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 2011, pp. 2003–2010.
- [63] Mustafa Misir et al. "An intelligent hyper-heuristic framework for chesc 2011". In: *International Conference on Learning and Intelligent Optimization*. Springer. 2012, pp. 461–466.
- [64] David E Moriarty, Alan C Schultz, and John J Grefenstette. "Evolutionary algorithms for reinforcement learning". In: *Journal of Artificial Intelligence Research* 11 (1999), pp. 241–276.
- [65] Gabriela Ochoa et al. "Hyflex: A benchmark framework for cross-domain heuristic search". In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer. 2012, pp. 136–147.
- [66] Randal S Olson and Jason H Moore. "TPOT: A tree-based pipeline optimization tool for automating machine learning". In: *Automated Machine Learning*. Springer, 2019, pp. 151–160.
- [67] Federico Pagnozzi and Thomas Stützle. "Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems". In: *European journal of operational research* 276.2 (2019), pp. 409–421.
- [68] Judea Pearl. "Intelligent search strategies for computer problem solving". In: *Addison Wesley* (1984).
- [69] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [70] Nelishia Pillay and Derrick Becketdahl. "EvoHyp-a Java toolkit for evolutionary algorithm hyper-heuristics". In: *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2017, pp. 2706–2713.
- [71] Dmytro Pukhkaiev and Uwe Aßmann. "Parameter Tuning for Self-optimizing Software at Scale". In: *arXiv preprint arXiv:1909.03814* (2019).
- [72] Dmytro Pukhkaiev and Sebastian Götz. "BRISE: Energy-Efficient Benchmark Reduction". In: *Proceedings of the 6th International Workshop on Green and Sustainable Software*. GREENS '18. Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 23–30.
- [73] Patricia Ryser-Welch and Julian F Miller. "A review of hyper-heuristic frameworks". In: *Proceedings of the Evo20 Workshop, AISB*. Vol. 2014. 2014.

- [74] Kumara Sastry, David Goldberg, and Graham Kendall. "Genetic algorithms". In: *Search methodologies*. Springer, 2005, pp. 97–125.
- [75] Bobak Shahriari et al. "Taking the human out of the loop: A review of Bayesian optimization". In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.
- [76] Jim Smith. "Self adaptation in evolutionary algorithms". PhD thesis. 2020.
- [77] Kenneth Sörensen, Marc Sevaux, and Fred Glover. "A History of Metaheuristics". In: *Handbook of Heuristics* to appear (Jan. 2017).
- [78] Jerry Swan, Ender Özcan, and Graham Kendall. "Hyperion—a recursive hyper-heuristic framework". In: *International Conference on Learning and Intelligent Optimization*. Springer. 2011, pp. 616–630.
- [79] Michael Syrjakow and Helena Szczerbicka. "Efficient parameter optimization based on combination of direct global and local search methods". In: *Evolutionary Algorithms*. Springer. 1999, pp. 227–249.
- [80] Chris Thornton et al. "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, pp. 847–855.
- [81] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.
- [82] Edward Tsang and Chris Voudouris. "Fast local search and guided local search and their application to British Telecom's workforce scheduling problem". In: *Operations Research Letters* 20.3 (1997), pp. 119–127.
- [83] Gönül Uludağ et al. "A hybrid multi-population framework for dynamic environments combining online and offline learning". In: *Soft Computing* 17.12 (2013), pp. 2327–2348.
- [84] Enrique Urrea Coloma et al. "hMod: A software framework for assembling highly detailed heuristics algorithms". In: *Software Practice and Experience* 2019 (Mar. 2019), pp. 1–24.
- [85] Christos Voudouris and Edward Tsang. "Guided local search and its application to the traveling salesman problem". In: *European journal of operational research* 113.2 (1999), pp. 469–499.
- [86] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.
- [87] Gerhard J Woeginger. "Exact algorithms for NP-hard problems: A survey". In: *Combinatorial optimization—eureka, you shrink!* Springer, 2003, pp. 185–207.
- [88] David H Wolpert and William G Macready. "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

Statement of authorship

I hereby certify that I have authored this Master Thesis entitled *From Parameter Tuning to Dynamic Heuristic Selection* independently and without undue assistance from third parties. No other than the resources and references indicated in this thesis have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present thesis. I am aware that violations of this declaration may lead to subsequent withdrawal of the degree.

Dresden, 5th April 2020

Yevhenii Semendiak