TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Software- und Multimediatechnik, Lehrstuhl für Softwaretechnologie

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: 30th March 2020

Technische Universität Dresden/Fakultät Informatik

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

Verteiler: 1 x SCIS, 1x HSL, 1x Betreuer, 1x Student

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 9
2.1.2 Optimization Problem Solvers 11
2.2 Heuristic Solvers for Optimization Problems 14
2.2.1 Simple Heuristics 15
2.2.2 Meta-Heuristics 15
2.2.3 Hybrid-Heuristics 19
2.2.4 No Free Lunch Theorem 20
2.2.5 Hyper-Heuristics 20
2.2.6 Conclusion on Heuristic Solvers 24
2.3 Parameter Tuning as an Optimization Problem 24
2.3.1 Approaches for Parameter Tuning 25
2.3.2 Systems for Model-Based Parameter Tuning 26
2.4 Parameter control as an Optimization Problem 30
2.4.1 Parameter Control Definition 30
2.4.2 Examples and Reported Impact 30
2.5 Conclusion 30

3 Concept Description 31
3.1 Search Space 31
3.2 Prediction Process 31
3.3 Low Level Heuristics 31
3.4 Conclusion of concept 32

4 Implementation Details 33
4.1 Hyper-Heuristics Code Base Selection 33
4.1.1 Requirements 33
4.1.2 Parameter Tuning Frameworks 33
4.1.3 Conclusion 33

4.2 Search Space . . . 34
4.2.1 Base Version Description . . . 34
4.2.2 Implementation . . . 34
4.3 Prediction Logic . . . 34
4.3.1 Base Version Description . . . 34
4.3.2 Predictor . . . 34
4.3.3 Prediction Models . . . 34
4.4 Data Preprocessing . . . 34
4.4.1 Heterogeneous Data . . . 34
4.4.2 Base Version Description . . . 34
4.4.3 Wrapper for Scikit-learn Preprocessors . . . 35
4.5 Low Level Heuristics . . . 35
4.5.1 Requirements . . . 35
4.5.2 Code Base Selection . . . 35
4.5.3 Scope of work analysis . . . 35
4.6 Conclusion . . . 35

5 Evaluation 36
5.1 Evaluation Plan . . . 36
5.1.1 Optimization Problems Definition . . . 36
5.1.2 Hyper-Heuristic Settings . . . 36
5.1.3 Selected for Evaluation Hyper-Heuristic Settings . . . 37
5.2 Results Discussion . . . 37
5.2.1 Baseline Evaluation . . . 37
5.2.2 Hyper-Heuristic With Random Switching of Low Level Heuristics 37
5.2.3 Parameter Control . . . 37
5.2.4 Selection Only Hyper-Heuristic . . . 37
5.2.5 Selection Hyper-Heuristic with Parameter Control . . . 37
5.3 Conclusion . . . 37

6 Conclusion 38

7 Future work 39

Bibliography 40

List of Figures

2.1 Target System 10
2.2 Meta-heuristics Classification 17
2.3 Evolutionary Algorithm Control Flow 18
2.4 Hyper-Heuristics 21
2.5 Automated Parameter Tuning Approaches[1] 25

[1] Graphics from [35]

List of Tables

5.1 System settings for benchmark 36

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 [**surv:kerschke2019automated**] → exploration-exploitation balance
- no-free-lunch (NFL) theorem [60] → motivation of the thesis

thesis motivation The most related research field is Hyper-heuristics optimizations [11], that are designed to intelligently choose the right low lewel 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 [***surv:kerschke2019automated***])*

Analysis of existing studies in field of parameter control and algorithm configuration problems *(find a problem definition)* [37]

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 it 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 the base knowledge in field of Optimization Problems and the process of their solving. The reader who is an expert 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* seems like two different names for one thing, we encourage you to read this chapter carefully. We highly recommend for everyone to refresh the knowledge 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.

In this chapter we...

2.1 Optimization Problems and their Solvers

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 [27].

In this thesis we focus on the Optimization Problems — a special case of the Search Problems.

A lot of conducted studies in this field have tried to formalize the concept of OP, but the underlying notion such a vast that it is almost impossible to exclude the application domain from the definition. The description of every possible Optimization Problem and all approaches for solving it are out of the scope of this thesis. However, a birds-eye view should be presented in order to make sure that reader is familiar with all notions used through this thesis.

In [1, 6, 24] 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 subject of the Optimization. In general, it could be imagined as the Target System (TS) displayed on picture 2.1. Analytically it could be represented as the function $Y = f(X)$. Informally it accepts the information with its *inputs* $\mathbf{X}$ sometimes

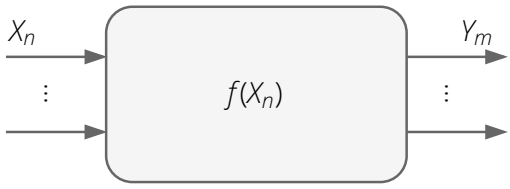


Figure 2.1 Target System

also called variables or parameters, performs a *Task* and produces the result on its *outputs* Y.

Pair of *X* and respective *Y* form a *Solution* for Computation Problem. All possible inputs *X* form a *Search Space*, while all outcomes *Y* form an *Objective Space*. The Solution could also be characterized by the *objective* value(s) — a quantitative measure of TS performance that we minimize or maximize. We could obtain those value(s) directly by reading the *Y*, or indirectly for instance, noting the time TS took to produce the output *Y* for given *X*. The Solution objective value(s) form object(s) of Optimization. For the sake of simplicity we here use *Y*, *outputs*, *objectives* and *X*, variables, parameters(?)interchangeably.

Next, let us highlight the Target System characteristics. Among mentioned in [1, 6, 24] we found those the most important:

- Input data types of *X* 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.
- Constrains are functional dependencies that describe the relationships among inputs and defile the allowable values for them.
- Amount of knowledge TS exposes about the dependencies between $X \rightarrow Y$ or objective values. With respect to this knowledge, the Optimization could be *White Box* — the TS exposes it internals fully, so it is even possible to derive the algebraic model of TS. *Black Box* — the exposed knowledge is mostly negligible.
- Dependencies randomness One of possible challenges, while obtaining the knowledge about TS is uncertainty of output. Ideal case is the *deterministic* dependency between *X* and *Y*, however in most of real-world challenges engineers tackle with the *stochastic* systems whose output is affected by random processes.
- Cost of evaluation is the amount of resources (computational, time, money, etc.) TS will spend to obtain the result for particular input. It varies from very cheap if the TS is a simple algebraic formula and Task is to evaluate it, to very expensive if the TS is a complex Neuron Network and the Task is to train it on data.
- Number of objectives could be either *Single*, or *Multiple*. According to the number of objectives, the result of optimization will be either single Solution, or set of non-dominated (Pareto-optimal) Solutions [18].

Combining different characteristic, one could obtain broad range of Optimization Problem types.

In this thesis we tackle such a real life problems as bin packing, job-shop scheduling or vehicle routing. The mentioned above problems has been shown to impose *NP – complete* (meaning that they are both *NP* and *NP – hard*) computational complexity [26].

As an example, let's grasp these characteristics for Traveling Salesman Problem (TSP) [2] — an instance of vehicle routing problem and one of the most studied combinatorial OP, yet still remaining one of the most challenging (here we consider deterministic, symmetric TSP). The informal definition of TSP is as follows: 'Given a set of cities and the distances between each of them, what is the shortest path to visit each city once and return to the origin city?'. The input data (path) is a vector of city indexes, and those the type is a non-negative integers *0, 1, 2...*. 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. The TSP distance (or cost) matrix here plays role of Target System, clearly that this TS exposes all internal knowledge and those it is the white box. Since the cost matrix is fixed and not changing, the TS is considered to be deterministic, cost for two identical paths are always the same (although there exist Dynamic TSP where the cost matrix changes while computing the path cost to reflect a real-time traffic information updates while traveling [14]). It is extremely cheap to compute a cost for given path using cost matrix, those overall Solution evaluation in this TS is cheap. Since we are optimizing only the route distance, it is a Single objective OP.

2.1.2 Optimization Problem Solvers

Any Optimization Problem could be solved by an exhaustive search. But when the problem size significantly increase, the amount of time needed for an exhaustive search becomes infeasible and in most cases even relatively small problem instances could not be solved by an enumeration.

Here different techniques come into play, but the provided by Target System characteristics of Optimization Problem 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 described using Integer Linear Programming

Yevhenii: ref

approaches, or heuristics

Yevhenii: ref

. If this TS turned out to be a black box, the ILP approaches are not applicable and one should consider using heuristics [6].

Again, there exist a lot of different facets for OP Solvers classification, however they are a subject of surveying works. Here as the point of interest we decided to highlight two of them.

From the perspective of solution quality:

- **Exact** Solvers are those algorithms that always provide an optimal Solution for OP.
- **Approximate** Solvers produce a sub-optimal output with guarantee in quality (some order of distance to the optimal solution).
- **Heuristics** Solvers do not give any worst-case guarantee for the final result quality.

From the perspective of solution availability:

- Algorithms that expose the Solution **at the end** of their run.
- In opposite, **anytime** algorithms designed to improve the solution quality step-by-step while solving the OP and those, intermediate results are naturally accessible.

Each if this algorithm families has own advantages and disadvantages in comparison to other, and if the property of solution availability is clear, the solution quality faced require more detailed description.

Solution Quality Classes

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

For some OP it is possible to develop an algorithm that is much faster than exhaustive search — it runs in super-polynomial time providing an optimal solution. As it stated in [59], 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 an NP-complete problem.

By the definition in [25], the objective of an exact algorithm is to perform better (in terms of running time) than exhaustive search. In both works [59] author had enumerated the main techniques for exact algorithms designing each of which enhance this 'better' independently. A brief explanation of them will help to refresh the knowledge.

- Branching and bounding techniques when applied to origin problem, split the search space of all possible solutions (e.g. exhaustive search space) to a set of smaller sub-spaces (more formally, branching the search tree into subtrees). This is done with an intent to later prove that some sub-spaces never lead to an optimal solution and those could be ignored in order to speed-up the search.
- Dynamic programming across the Subsets techniques in some sort could be combined with the mentioned above branching techniques. After forming the Search Space subsets (branches), the dynamic programming attempts to derive solutions for smaller subsets and combine them into solutions for lager subsets unless finally derive a solution for original search space.
- 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 reduce the running time. A toy example from [59] 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 a set). The exhaustive search will enumerate all $x - y$ pairs in $O(k^2)$ time. But one could first preprocess the data by sorting it, after that use bisection search repeatedly in this sorted array and search for k values $S - y_i$, the overall time complexity becomes $O(k \log(k))$.

Approximate Solvers. When the OP cannot be solved to optimal in polynomial time, people start thinking in alternative solutions and mostly relax their requirements. Approximate algorithms are representatives of the theoretical computer science field that have been created in order to tackle the computationally difficult white box OP.

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 [58]. The worst case results quality guarantee is crucial in design of approximation algorithms and involves mathematical proofs.

One may ask a question "How do these algorithms guarantee on quality, if the optimal solution is unknown ahead?" Certainly it sounds contradictive 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.

In [58] authors stated several techniques of an approximate solvers design. 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 LP. Later fractional solution for the LP should be rounded to obtain a feasible solution for the ILP. Different rounding strategies define separate technique for approximate solvers [58]:

- Deterministic rounding follows predefined ahead strategy for rounding.
- In Randomized rounding the algorithm will do a round-up of each fractional solution value to integer uniformly.

Another technique in contrast to rounding requires building a *dual linear program* for LP. This technique utilizes a *weak* and *strong duality* properties of dual linear program to derive the distance of approximate from an optimal solution. Other properties of dual linear program form a basis for Primal-dual algorithms. They start with a dual feasible solution and use dual information to derive a solution (possible infeasible) for primal linear program. If the *primal* solution is infeasible, algorithm modifies dual solution to increase the values of the dual objective function [58].

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

Heuristics could be defied as rules of thumb, or strategies to use available from TS and obtained solution information to control a problem-solving process [48]. Scientists draw the inspiration for heuristics creation from all aspects of our being — from observations of how humans tackle problems using intuition to mechanisms discovered in nature.

As well as in previous approaches, there exists a lot of facets for heuristic approaches classification. However, from the *level of generality* perspective exist:

- Simple heuristics are algorithms, specifically designed to tackle concrete problem. They use the domain knowledge from Optimization Problem to gain a performance. Simple heuristics do not provide any mechanisms to escape a local optimum and those could be easily trapped to it [48].
- Meta-heuristics are high-level heuristics that do not require problem domain knowledge and those could be applied to broad range of OPs. Often they are nature-inspired and comprise mechanisms to escape local optima and also converge slower than simple heuristics [5].
- Hybrid-heuristics arise combinations of two or more meta-heuristics. It could be imagined as a combination of recipes from the cook book to create something new and probably better, merging the best characteristics.
- Hyper-heuristics is a heuristic that operates not on the Search Space constructed for the OP, but on a set of low-level heuristics, used to solve the Optimization Problem. The research and experiments have shown that some meta-heuristics perform better for some types of problems, but poorly for other. In addition, it could happen so that for different instances of the same problem, various meta-heuristics provide unexpected performance metrics. Even in different stages of the problem solving process the dominance of one heuristic over another could change. Here comes hyper-heuristics to intelligently pick suitable meta-heuristics to solve a problem [11].

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

Selecting Best Suited Solver

An old engineering slogan says, "Fast, Cheap or Good? Choose two."

And here we should make a decision, which way to follow. In one hand, we have an exact solver for the Optimization Problems. As we mentioned above, it guarantees to derive an optimal solution, always. Today, tomorrow or in next century, but an exact solver will find it. The only thing we need is simply (or not) construct an exact algorithm for our specific problem. This approach definitely provides us *good*, say the best, quality of final solution, however it sacrifices simplicity and speed in building a solver and solving the problem.

On the other hand we have an approximate solver. It does not guarantee to find an optimal solution, but instead reasonably good one. The required effort for constructing an algorithm and proving its preciseness remains the same as for exact solvers, from our perspective. Nevertheless, this approach beats the previous one in terms of speed in problem solving, sacrificing a reasonably small amount of the result quality. Sounds like a good deal.

Last but not least, remains bright and shining heuristic. It is super-fast in comparison to previous two approaches in problem solving. It is much easier to apply for your specific problem — no need to build complex mathematical models or prove theorems. However, the biggest flaw in this approach is that it does not guarantee to provide an optimal solution at all and those, one should consider use it up to own risk.

Modern world is highly dynamic, in business survive those who faster and stronger. In most cases former plays settle role for success and great products build iteratively, enhancing existing solution step-by-step and throwing away unlucky decisions quickly. As we mentioned in 2.1.1 this thesis is dedicated to facing a real-life problems such as TSP. The problems showed to be *NP – complete* that is why we are not allowed to apply exact solvers, only approximate and heuristics left. In both cases we are sacrificing a solution optimallity, although in different quantities, but the heuristic algorithms repay in time-to develop and getting first results. It motivates us to follow the heuristic approach through 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 in order to select the best suited heuristics class.

2.2 Heuristic Solvers for Optimization Problems

Before diving into description of heuristics and their concrete examples, it is worth to scratch a use-case. We base our descriptions on mentioned in section 2.1.1 Traveling Salesman Problem (TSP) [2], which informal definition sounds as: 'Given a set of cities and the distances between each of them, what is the shortest path to visit each city once and return to the origin city?'. The input data X to our heuristics will be the distance matrix (or coordinates to build this matrix) between cities, as an output from heuristics we expect to get the sequence of cities, describing the route plan.

In general, heuristics when applied to particular problem do not use the gradient or Hessian matrix of the objective function for optimizations [9].

Most heuristic approaches imply and use following concepts:

- Neighborhood defines the set of Solutions, which could be derived performing one step of heuristic search.
- Iteration could be defined as the action (or set of actions) done over Solution in order to derive a new, hopefully better Solution.

- **Exploration** (diversification) is the process of discovering unvisited and high quality parts of the Search Space.
- **Exploitation** (intensification) is the usage of already accumulated information about the Search Space to derive a new one, but similar to existing Solutions.

2.2.1 Simple Heuristics

As we mentioned above, the simple heuristics are domain dependent algorithms, designed to solve a particular problem. Since each application is highly defined by the concrete OP instance, it is hard to distinguish commonalities among simple heuristics except of only domain knowledge utilization.

Two main types of simple heuristics were distinguished in [12]. The first is a *perturbative*, or *Local Search* heuristics which operates on completely created Solutions. The prominent example of Local Search is a *Hill Climbing*. This approach plays a central role in many high-order heuristics. The second is a *constructive* heuristic which step-by-step mature partial candidate Solution.

A concrete example of Local Search is *Greedy Algorithm*, also known as *Best Improvement Local Search*. When applied to Traveling Salesman Problem, it tackles the path construction by simply accepts the next closest city from currently discovered. In general, the Greedy Algorithm follows the logic of making a sequence of locally optimal decisions.

Other instance of Local Search is *First Improvement* Local Search [57]. This algorithm accepts a better Solution as soon as it finds it. The advantage of this methodology over the Greedy Algorithm is the velocity of Search Space traversing, since to perform a move, Best Improvement first should evaluate entire Neighborhood, which in some cases could be enormously huge.

Indeed, the use of simple local search heuristics might not lead to a globally optimal solution, since the optimization result is fully defined be the starting point, but the advantage is a simplicity in implementation [58].

Yevhenii: need to add more info, probably structurize somehow

2.2.2 Meta-Heuristics

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

The prefix *meta* indicates that these algorithms are heuristics of a *higher level* when compared to a simple problem specific 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. Also, the optimizer could contain stochastic components, what gives it an ability to escape from local optimum. However,it also means that the output is stochastic and could not guarantee the result preciseness [9].

The success of meta-heuristic on a given OP depends on the balance between exploration and exploitation. If there is a strong bias towards diversification, the solving process could naturally skip a good solution while performing a huge steps over the search space, but when the intensification dominating, the process will quickly converge to local optima. A simple heuristic approaches suffer from high exploitation dominance. In general, the difference among existing meta-heuristics laid in a particular

way how they try to achieve this balance, but the common property is that most of them are inspired by processes in nature — physics, biology, ethology or evolution.

Classification

The research of meta-heuristics field arise even before 1940s, when they had being used but not formally studied. In the period of between 1940 and 1980s appear first formal studies and later till 2000 the field of meta-heuristics appears in wide and numbers of methods were proposed. The period from 2000 and until now in [52] authors refer as the time of framework growth. It is now a time, where meta-heuristics widely appear as frameworks, requiring a domain specific adaptation and providing a reusable core.

The development of novel methods has slowed down, the research community began to organize these algorithms and many classification facets were distinguished. As an example, the research conducted by [7] highlights following characteristics:

- The **method of walk-through** could be either trajectory based or discontinuous. The former one corresponds a closed walk on the neighborhood where Simulated Annealing, Tabu Search or Local Search are typical examples. The later one allows large jumps in the search space where examples are Variable Neighborhood Search, Lin-Kernighan Heuristic for the TSP [39].
- By the **number of concurrent solutions** we distinguish *single-point* and *population-based* approaches with Tabu Search, Simulated Annealing, Iterated Local Search examples of former and Evolutionary Algorithms, Ant Colony Optimization, Particle Swarm Optimization are instances of later.
- From the **memory usage** perspective highlighted those which *does utilize memory* and *memory-less* approaches. The Tabu Search explicitly use memory in forms of tabu lists, but 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.

Picture 2.2 illustrates the summarized classification including some other characteristics and well-known meta-heuristic samples [15].

Examples

We shortly describe some prominent and widely used meta-heuristics, since later we use them as the LLH in developed 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 these methods is as following. 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 object of optimization for given subject TS) [22]. Tree basic actions are defined as operators of EAs: the *recombination* that is applied to selected candidates Solutions (parents) among available in population to produce new ones (children); *mutation* occurs in one candidate to turn it into a new Solution. Applying both operators on the parents create a set of new Solutions — the offspring, whose results evaluated using TS. After that, the

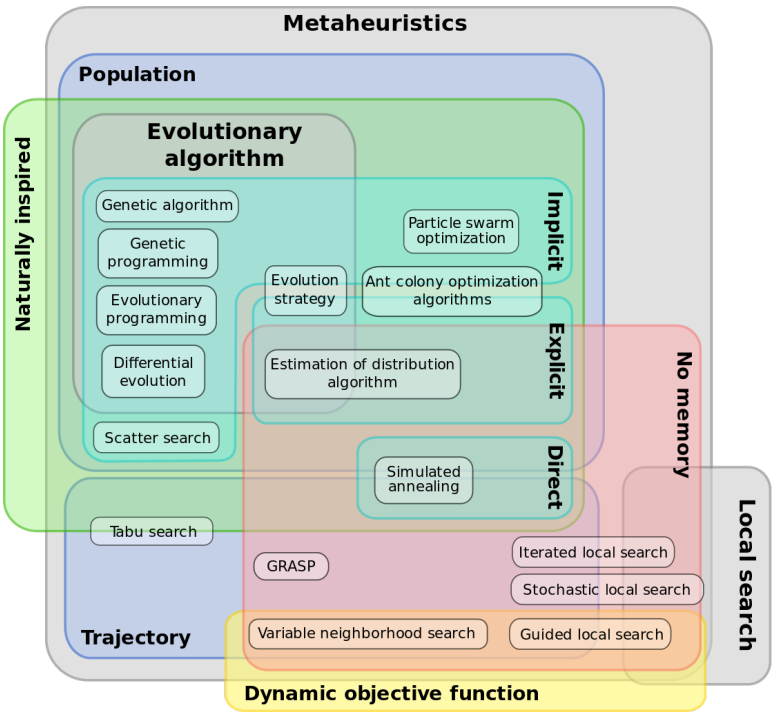


Figure 2.2 Meta-heuristics Classification

selection operator applied among available Solutions (parents and offspring) to keep the population size within defined boundaries. This process is repeatedly iterated until some termination criteria fulfilled, for instance maximal number of iteration reached, amount of TS evaluations exceed, or Solution with required quality found. The work-flow of EA depicted on picture 2.3.

Genetic Algorithm (GA). It is the first association coming into mind when you hear words 'Evolutionary Algorithms'. GA traditionally has a fixed work-flow: with given initial population of μ usually randomly sampled individuals, the parent selection operator shuffles initial set to create a random pairs of parents, after that the crossover operator is applied to each pair with probability p_c to produce children. Then newly created Solutions individually undergoes mutation operator with independent probability p_m. Resulting offspring perform tournament within selection operator and μ survivals replace current population [21]. Distinguishable characteristics of simple GA are following: Solution representation in form of bit-strings, one-point crossover and bit-flip usage as the recombination and mutation operators respectively and a generational selection operator (survive only children).

Evolution Strategy (ES). In contradiction to GA, Evolution Strategy algorithms are working in a vector space of Solution representation and distinguish μ individuals population size and λ offspring generated in one iteration. They induce a very useful feature of *self-adaptation*: changing the mutation step sizes depending on control parameters. The self-adaptive information (which is related entirely to EA, but not to OP under consideration) is appended to the individual's chromosome. While the general work-flow for all EAs remains the same, underlying operators are changed. Here, parent selection operator take a whole population into consideration, the recombination scheme could involve more than two parents to create one child. To construct a child, recombination operator adds alleles from parents in two possible ways: with *uniform*

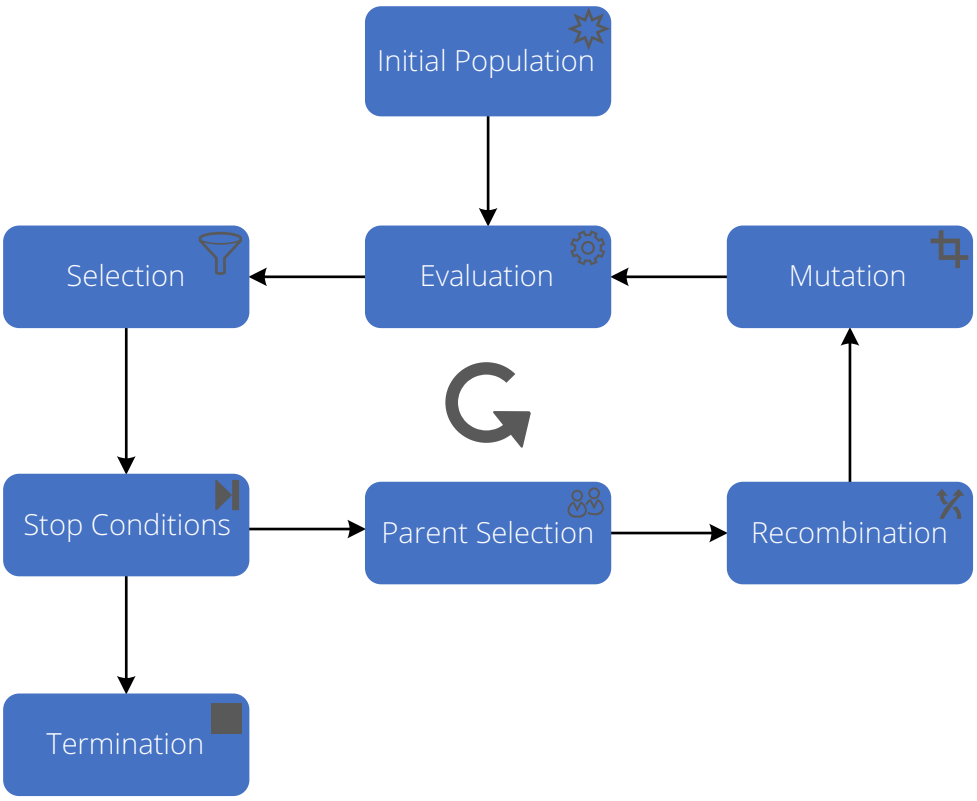


Figure 2.3 Evolutionary Algorithm Control Flow

probability for each parent (discrete recombination), or averaging the values of alleles (intermediate recombination). There are two general selection schemes that are used is such algorithms: (μ, λ) which discards all parents and selecting only among offspring highly enriching exploration, and $(\mu + \lambda)$ which includes predecessor Solutions into selection which also known as *elitist selection* [21].

Simulated Annealing (SA). This meta-heuristic is inspired by the annealing technique used in the metallurgy to obtain 'well-ordered' solid state of metal, imposing a minimal internal energy and avoiding semi-stable structures, characterized by local energy minimums. The search process here 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^* selected within the neighborhood $N(S)$ of current S and the selection criteria evaluated based on S^* quality and temperature parameter T. 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 temperature: $p(T, f(S^*), f(S)) = \exp(-\frac{f(S^*)-f(S)}{T})$ for minimization OP and $p(T, f(S^*), f(S)) = \exp(-\frac{f(S)-f(S^*)}{T})$. At each iteration the temperature parameter T is decreased following *Annealing Schedule* also called as *Cooling Rate*: linearly, inverse logarithmic, exponentially [9]. The weak side here is that the 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, refer [32] and [17] respectively.

2.2.3 Hybrid-Heuristics

The hybridization of different systems constantly provides a positive effect — you take advantages of one system and merge them with the strong sides from other those getting the best from both of them. The same could be applied for the heuristics. Imagine you have two algorithms biased towards exploration and exploitation respectively. If you use them separately, the expecting results in most cases may be way far from optimal as the outcome of disrupted diversification-intensification balance. But with merging them into say stages of hybrid heuristic, one will obtain both advantages of finding a good quality results and escaping local optima. Most of available hybridization are done exactly with this idea — *staging combination* two heuristics, one exploration and second exploitation suited for getting outperforming hybrid.

Examples

The methods to construct hybrid mostly defined by the undertaken heuristics and those, to the best of our knowledge, could not be generalized and classified well, except a *staging* approach, when the output of one algorithm is used as initial state of other, is broadly used. Instead, 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.

Guided Local Search (*GLS*) + Fast Local Search (*FLS*) [55]. It is an example of repeatedly applying two heuristics in sequence (staging) and passing the output from one to second one. The main focus of *GLS* here is on the Search Space exploration and process guidance using incubated information. In some sort, *GLS* is closely related to the *Frequency-based memory* used in Tabu Search. In runtime, *GLS* modifies the cost function of the problem to include penalties and passes this modified cost function to local search procedure. The penalties form memory that describe a local optimum solution and guide the process out of it. A local search procedure then carried out by *FLS* algorithm. The main feature of *FLS* is the ability to quickly traverse a neighborhood. It is done by braking it 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* appears in the local optimum, passes back control to *GLS* and iteration repeats.

Direct Global + Local Search [54]. As stated in the name this hybridization combines global and local optimization strategies into two-stages: stochastic global pre-optimization and deterministic local fine-optimization. For global optimizations authors apply one of two well-known meta-heuristics — Genetic Algorithm, or Simulated Annealing described earlier in section 2.2.2 with Meta-heuristics examples. The transition from Global to Local search happens when the predefined conditions are met, for instance when the number of Target System (goal function) evaluations exceeds a boundary or when no distinguishable improvement was done. The Pattern Search [29] algorithm also known as direct, derivative-free, or black-box search plays role of Local Search heuristic in this combination. Hybrid-heuristic terminates when Pattern Search converges.

Simulated Annealing + Local Search [42]. After brief explanation of previous two hybrids, an observant reader might make a guess what happens in this particular case, and he will be completely right! 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 (authors called this action a 'kick', referring a dramatic change of current position within a Search Space). After this, the Local Search heuristic applies to intensify obtained Solution. When the local optimum reached, the control flow returned to the Simulated Annealing for acceptance criteria evaluation in order to accept or reject a new Solution, what concludes an iteration.

EMILI **[47]** 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 which define a search of an optimal sequence of steps for product creation within a workshop. Here authors have implemented algorithmic- and problem-specific building blocks, defined grammar-based rules for those building blocks composition and use an automatic algorithm configuration tool *IRACE* [41] to find a high performing algorithm configurations for problem solving. The work-flow of *EMILI* could be described in three steps: (1) adaptation of rules to specific representations type of PFSP problem objectives (either Makespan, Sum completion times, Total tardiness), (2) generate a set of possible hybrid heuristics for each of PFSP type and (3) apply iterated racing algorithm implemented in *IRACE* to select the best performing hybrid for specific problem type.

From our perspective, described approach of automatic algorithm generation is very close to construction Hyper-Heuristics strategies with off-line learning described in section 2.2.5, however authorized to change the system class (from hybrid- to hyper-heuristic) defined by *EMILI* authors.

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 spend effort and develop new algorithms, instead of using existing?" And the answer to this question is quite simple — the perfect algorithm suited for all OP that will do not exist and can not exist. All algorithms that search for optimal parameters of a Target System perform exactly the same, when the results are averaged over all possible Target Systems. 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** (NFLT) [60].

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 the specific approach, and analyze it behavior with respect to another in problem solving process. Here simple heuristics and meta-heuristics are out of competition, since if you solved the Optimization Problem once, you wouldn't optimize a second time. Here come **Hyper-Heuristics**. We will proceed with their description and how they deal with the NFLT consequences in following section, to not switch the thesis.

2.2.5 Hyper-Heuristics

Lots of state-of-art heuristics and meta-heuristics are developed in a very domain-dependent way, say they use the domain knowledge too intensively to be broadly reused. 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

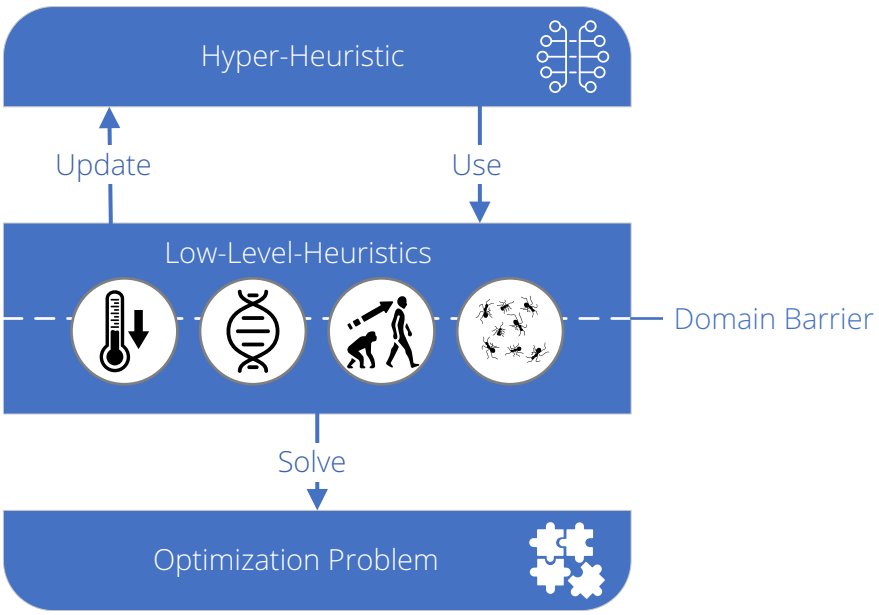


Figure 2.4 Hyper-Heuristics[a]

[a]Icons from thenounproject.com

approach of using some High-Level-(Meta-)Heuristics (HLH) to choose other Low-Level-(Meta-)Heuristics (LLH) and use it to solve the OP in hand. Indeed, a combination of different low-level heuristics produced better results than if hey were applied separately [19]. It can be explained by the nature of search process and how it evolves in time. When you apply heuristic, it sooner or later converge to some extreme point, hopefully global optimum, since it is 'blind' to those not visited regions in the Search Space. But 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) rapidly. However, it is not possible to predict how one LLH will behave in every stage of the search process in comparison to another, here HLH comes to help. In [45] authors made infer that Hyper-Heuristics can be viewed as a form of Reinforcement Learning, which is sounds logically.

The new concept which implicitly was used in Meta-Heuristics, but explicitly pointed out in Hyper-Heuristics is the **Domain Barrier**. As we told previously, HH do not directly tackle an OP, they use LLH instead. This means, that HH are usually unaware of the domain details such as what are those data types, their relationship, etc. within a domain, but rather encapsulates this information in LLHs and those could be used to broader range of Optimization Problems as it is illustrated on picture 2.4.

Classification

Although, the research in Hyper-Heuristics field actively ongoing, a lot of different instances were already created and some trials to organize approaches were conducted [12, 19, 34, 51]. Researchers in their surveys classify HHs by different characteristics, some of them overlap, but it also happens that intriguing parameters distinguished were not highlighted in other works.

In this section we present some (but not all) facets of Hyper-Heuristics classification to better justify the goal of this thesis.

We begin with the two broadest classes, which differentiate HH *routine* or also called

as *nature of High-Level-Heuristic Search Space* [12, 13, 19]. The first class is the Hyper-Heuristics to **select** Low-Level-Heuristic, in other words **Selection Hyper-Heuristic**. In previous sections we were implicitly referring to this class, while talking about Hyper-Heuristic approaches in general. These algorithms operate in the Search Space defined by simple Low-Level-Heuristics that solve Optimization Problem. The task of HLH is to pick the best suited LLH based on available prior knowledge and apply it to OP underway. The second class is the Hyper-Heuristics to **construct** or generate LLH by using the atomic components of other heuristics as Lego bricks and following some predefined receipt. These approaches could lead to creating new and unforeseen heuristics that are expected to reveal good performance [12] while solving the problem in hand.

Next, the distinction in *nature of Low-Level-Heuristics Search Space*, in other words how do the LLH derive Solutions for the OP in hand [12, 13, 19], either by **constructing** a Solution each time from scratch, or by **perturbation** of already existing one.

The other broadly used characteristic is the *use of memory*. From this perspective we distinguish Hyper-Heuristics in which the learning happens on-line, off-line or learning mechanisms are not present at all [12, 51].

- In **on-line** case, the HH derives an information, used to select among LLH, while those LLH are solving a problem.
- In **off-line** case, the learning happens before solving an Optimization Problem. Here one should first train an HH using other, but homogeneous to current problems. After this preparation step, the HH could be applied to problem in hand.
- There exist also **mixed** cases, where learning happens both on-line and off-line. Definitely it is a promising research direction, despite high dependency on off-line phase.
- The last case here is an approach **without learning** mechanisms involved. Usually, these Hyper-Heuristics perform some sort of Random Search selection of LLH.

Yet another faced of Hyper-Heuristics classification is the way of assigning *hyper-parameters* (here we use parameters and hyper-parameter concepts interchangeably) for LLHs, or their components [19]. We analyzed surveys and find out that some researchers do not explicitly differentiate approches with respect to nature of parameter settings [12, 13, 51], while other do [19]:

- In **static** assignment, the underlying heuristics use provided beforehand (usually default) hyper-parameters and do not change them while solving the problem in hand.
- The **dynamic** case uses some kind of rule for parameters changing, specified in advance.
- There exist also an **adaptive** approach, in which HH assigns the parameters for LLH as the response to the learning process. In some sort, it is similar to the parameter control techniques used in Meta-heuristics.
- And finally, a **self-adaptive** approach where underlying LLHs comprise *parameter control* techniques and those search for the best solution for OP and own parameter settings simultaneously.

For more detailed analysis, description, other classification facets and respective Hyper-Heuristic examples we encourage reader to look into [11, 12, 19, 34, 51] researches.

Examples

HyFlex [46] *Hyper-Heuristics Flexible Framework*. It is a software skeleton, created specifically to encourage 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), evaluation functions and a set of Low-Level-Heuristics. The benchmark sets as well as comparison to other existing HH is included to framework. The intent of *HyFlex* creators to provide these features was to enable others focus directly on High-Level-Heuristics implementation without need to challenge other minor needs and those bring clear comparison among HLH performance [46]. From the classification perspective, all derivatives from the *HyFlex* framework are Selection Hyper-Heuristics, however they utilize different learning approaches and hyper-parameter settings. Algorithms, built on top of *HyFlex* framework could be found in [19, 44, 51] or on the CHeSC 2011 challenge website[1].

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

- *Hyperion* [53] is a *TH* (recursive template and hook) framework aiming to extract information from OP search domain for identification of promising components.
- *hMod* [56] framework allows not only to construct algorithm components using predefined abstractions (such as *IterativeHeuristic*). In current development stage, developers of *hMod* are focusing on creation of mechanisms rather than providing a set pre-built heuristics.
- *EvoHyp* [50] focus on hyper-heuristics with evolutionary algorithms used as Low-Level-Heuritics. Here authors enable users of framework to construct both selection and generation HHs for both types construction and perturbation LLHs.

HITO [28] *Hyper-Heuristic for Integration and Test Order Problem*. It is an example of HH for selection of LLH. 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[20] using interchangeably Choice Function (in form of weighted linear equation) and Multi Armed Bandit based logic to yet again balance exploitation of good LLHs and exploration of new LLHs.

MCHH [43] *Markov Chain Hyper-Heuristic* 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 LLH using Markov Chains those guiding an LLH selection process. Applying on-line reinforcement learning techniques, this HH adapts transition weights in the Markov Chains constructed from all available LLHs those updating prior knowledge for LLH selection.

Auto-Sklearn [34] Although, the application of Machine Learning (ML) techniques is not a brand-new idea, there exist numbers of automated machine learning systems. They automatically choose the best suited algorithm, feature preprocessing methods for a new dataset at hand with an objective of algorithm's accuracy maximization.

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

From perspective of Optimization Problem solving, automatic ML processes are quite similar to Hyper-Heuristics. They both operate on Search Space of algorithms which later applied to problem in hand with objective to find the best performing one. In particular, based on *Scikit – learn* framework [49] *Auto – Sklearn* system uses number of sklearn classifiers to process a data which in some sort similar to usage of LLHs to process a Search Space. Resulting problem, formally called *Combined Algorithm Selection and Hyper-parameter Optimization*problem (CASH) is then solved by hyper-parameter optimization framework *SMAC* [30]. We outline a definition of CASH problem as well as description of *SMAC* framework in 2.3 and 2.3.2 respectively.

2.2.6 Conclusion on Heuristic Solvers

To conclude our review on Heuristic approaches for Optimization Problems solving, 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 problem in hand. Usually, they are created to tackle a concrete problem in hand applying simple algorithmic approach. The simplicity of application and usually fast runtime is balanced by medium solution quality.

On the next level inhabit Meta-Heuristics. They could be compared with more sophisticated 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 a new environment (Optimization Problem), however some adaptations should be performed to understand the problem and parameter tuning to perform well.

Among with MHs, Hybrid-Heuristics do exist. It is nothing special here, they just took some survival abilities from several Meta-Heuristics hoping to outperform, however still requiring adaptation. In some cases this hybridization provides it advantage, but as time shows, they did not kick out MHs and reside together with their parents. Those we can conclude that the exposed balance between development effort and providing results quality not always assure users to apply them.

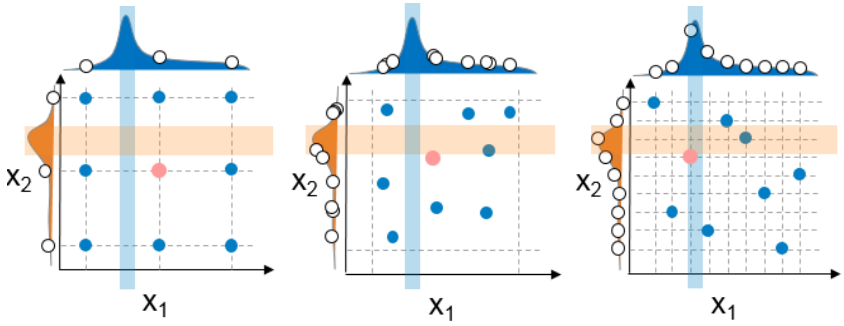
Finally, the chosen ones that lead the others, Hyper-Heuristics are inhabitants of the upper level generality. Operating by the other Heuristics, HHs analyze how good former are and definitely make use of this knowledge by solving concrete problem using the best suited Heuristic. Imposing such great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

2.3 Parameter Tuning as an Optimization Problem

Most of existing learning algorithms expose some parameter set, needed to be assigned before applying this algorithm to solve a problem. Modifying these parameters, one could change the system behavior and possible result.

When we are talking about parameter tuning problem, the following terms should be refined explicitly:

1. Target System (TS) is the subject of parameter tuning optimization problem. Simply, it is the system which parameters are tuned.
2. Parameter a.k.a. Hyper-Parameter (HP) is one of exposed by TS hooks for tuning. Should be described in terms of type and possible values.
3. Configuration is formed by unique combination of parameter values, required to run TS.



(a) Grid Search. (b) Random Search. (c) Learning Search.

Figure 2.5 Automated Parameter Tuning Approaches[a].

[a]Graphics from [35]

4. **Search Space** is formed by all possible Configurations for defined HPs.

In *machine learning* these parameters are called **Hyper-Parameters**. Note, that we are referring both terms *Parameters* and *Hyper – Parameters* (*HPs*) interchangeably through the thesis.

For instance, consider neuron network. Hyper-parameters in this case will specify the structure (number of hidden layers, units, etc.) and learning process (learning rate, regularization parameters, etc.) of network. Changing them will result in changes of system accuracy.

One of the most frequent optimization applications is **Parameter Tuning** — searching of hyper-parameter values to optimize some characteristic of system. When talking about our example, we could apply Parameter Tuning to maximize network accuracy. Considering simultaneously number of characteristics such as running time + accuracy results in Multi-Objective Parameter Tuning. When talking about heuristics, proper assignment of hyper-parameters has a great impact on exploration-exploitation balance and those on algorithm performance.

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.

2.3.1 Approaches for Parameter Tuning

Nowadays there exist wast number of ways to assign parameters for system. One of the simplest and error-prone ways is just trusting your (or someones else) intuition and using those parameters that seems to you more or less logic for particular system and problem instance. People quickly abandoned it in favor of automatic approaches, since novel computational capacities easily provide a possibility for it.

We briefly outline existing automated approaches illustrating each with a picture 2.5. Each graphic shows dependencies between parameters X_1 (x axis), X_2 (y axis) and optimization subject values (which we are maximizing here) along those axes.

Grid Search. It is a simple approach of solving the search problem by exchanging enumeration of possible solutions (parameter sets) for relaxed problem instance. This

relaxation is derived by user specifying finite number of possible values for each of hyper-parameters under consideration. After evaluating each possible configuration for system under estimation, algorithm reports the best found solution. Obviously, this approach could skip promising parts of search space as shown on picture 2.5a.

Random Search. This methodology relies on random (usually uniform) sampling of hyper-parameters and their evaluation on each iteration. At first 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 [3]. Compare visually the best configurations (highlighted in pink) found by grid (picture 2.5a) and random search (picture 2.5b) techniques.

Model Based Search. In most cases, the dependencies between tuned parameter values and optimization objective do exist, can be observed and utilized for hyper-parameter tuning by predicting which parameter combinations could produce a better results, those making guesses more precise. As it showed on picture 2.5c, after accumulating more information, learning algorithm will make more precise guesses, what in contrast to previously described model-free approaches is more preferable and robust. With this idea researchers started to build systems that (1) traverse the search space more efficiently and (2) could mimic dependencies between parameters and objective resulting in surrogate models. The later is some sort of enhancement used in combination with former enabling you to simulate evaluation of real system instead of expensive real probe.

2.3.2 Systems for Model-Based Parameter Tuning

The parameter tuning is an obligatory task for getting the maximum system performance and should be done at design time. Novel approaches for tuning are usually built in form of frameworks with exposed hooks for attaching system under estimation. Naturally, the Target System evaluations here supposed to be extremely costly since it requires actually executing of target algorithm, that is why algorithms in those frameworks are trying to utilize every single bit of information from evaluations, building surrogate models and using Bayesian optimization approaches for making each evaluation valuable.

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

- Conditional parameters support It exposed for user ability of tuning system to describe and tackle conditional dependencies between hyper-parameters. As an example imagine parameter that can take only some specific values, for instance *crossover type* in Genetic Algorithm could be *Partially Mapped Crossover (PMX)*, *Cycle Crossover (CX)*, etc. [36] Selecting concrete crossover type, one will also need to specify respective parameters for this crossover type. But it could turn out, that these parameters are illegal, when the other crossover type is selected. This type of dependency could be described by parent-child relationship, however other types of dependencies also exists.
- Parameter types support It is one of basic features required in system to be usable. Namely, Target System parameters could be not only numerical (more concretely integer or fractional), but also categorical in form of strings, boolean values, etc. Considering categorical data types, they could be either nominal or ordinal. Difference between categorical types lays in fact, that the latest depict

not only possible atomic values, but also order between them. For concrete example we discuss the Genetic Algorithm again with following parameters: population size (numerical integer in range [1... inf)), mutation probability (numerical fractional [0...1]), crossover type (categorical nominal *PMX*, *CX*). Indeed, population size could also be displayed as a set of finite values 10, 100, 1000 (categorical ordinal type).

- Extensibility This feature is crucial in cases, if one would like to try a new promising and intriguing learning algorithm, that was not included in parameter tuning system yet. In fact, one may need not only new learning algorithm, but 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 The simultaneous evaluation of multiple Configurations required for utilizing available computing resources that could scale horizontally.

IRACE [41]

The first system under investigation is an implementation if Iterated Racing Algorithm[8] in IRACE package for off-line hyper-parameters tuning. The underlying methodology consists three main steps: (1) using prior knowledge, sample new Configurations, (2) distinguish the best ones among sampled using racing algorithm and (3) update prior knowledge to bias next samplings in (1) towards better Configurations. Prior knowledge here represented as probability distribution for each tuned parameter independently (truncated normal and discrete distributions for numeric and categorical hyper-parameters respectively). During update step (3), the probability distributions are increased for those values, where the best Configurations were found.

Racing step (2) could be described as process of running Target System using Configuration under evaluation on a set of heterogeneous problem instances. While solving those instances, the statistically worse-performing Configurations are rejected and racing proceeds with remaining ones. This process continues until reaching required number of survivals or number of solved problem instances.

As it was mentioned above, IRACE supports various data types such as numerical and categorical as well as the possibility of conditions description. While the problem of data types solved by different distributions used, the conditional relationships are determined by the dependency graphs. During sampling (1), first non-conditional and only afterwards dependent parameters are sampled, if respective conditions are satisfied. The framework highly utilize racing algorithm for evaluations and *Friedman test* [16] or alternatively *paired t-test* for statistical analysis of parameters, those it is static in terms of variability and extensibility of learning mechanisms. In terms of parallel evaluations, the algorithm is doing well at the beginning of each racing step, however as the process proceeds less and less evaluations are executed in parallel those all available resources are utilized optimally at all steps of algorithm.

SMACv3 [30]

Sequential Model-based Algorithm Configuration (SMAC) is a system for parameters optimization developed by AutoML research group (here we review SMACv3). It is an extension introducing learning models to previously existing Random Online Aggressive Racing (ROAR) mechanism. Authors in their research showed that the machine learning mechanisms and regression models in particular could be applied not only for tuning

parameters, but also for optimizing any expensive black-box functions in general (we believe that the last holds also for the majority of other parameter tuning systems).

This system development was directed to tackle existing (at that point of time) 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) for facing the variance. A routine in Sequential Model-based Optimization is somehow similar to one implemented in IRACE and 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, at step (1) SMAC builds machine learning regression random forest [10]. The usage of regression decision trees (which form the forest) was motivated by known fact that they are known to fit well with categorical data. Later, at step (2) an Iterative Local Search (ILS) heuristic applied in combination with Bayesian optimization technique evaluating *Expected Improvement* (EI). ILS started on the best previously found Configurations and was used for sampling new promising ones while distinguish between them carried out by EI using regression model built at step (1). EI is large for those Configurations, with low predicted cost, or for those with high uncertainty those providing exploration-exploitation balance automatically [33].

Exposing such a great learning capabilities and using Expected Improvement technique that guarantees to converge the search process to global optimum given enough time, the major drawback in this system is lack of flexibility to include conditional dependencies between parameters into search space description. One of possible solutions here could be the use of conditional-aware neighborhood definition in ILS (currently it just carried out by sampling in Gaussian distribution). In fact, used Search Space representation framework ConfigSpace[40] is able to specify dependencies among hyper-parameters and verify the Configuration validity in terms of parameter conditions violation. 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. Obviously those cases are handled and broken Configurations are rejected, but in case of 'sparse' Search Spaces (where significant amount of parameter combinations are restricted by conditions) this approach could lead to ineffective sampling and predictive abilities of system will suffer greatly. Unfortunately, we did not find any officially published evaluations of such cases and could only make guesses based on own intuition and framework developers advises for SMACv3 application in such cases [2].

ROAR mechanism is the derivative from FocusedILS algorithm (solver in ParamILS parameter tuning framework [31]) 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, in principle, available parallel computation capabilities is another drawback of SMACv3 framework.

BOHB [23]

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 introduced by the same AutoML research group.

[2] Visit GitHub repository of SMACv3 for more info `https://github.com/automl/SMAC3/issues/403`

As it stated in name, the main routines here are carried out with mainly two algorithms: learning (1) and Configurations sampling (2) is done with Bayesian Optimization technique Tree Parzen Estimator (TPE) while Configurations evaluation and comparison carried out by HyperBand (HB) algorithm.

The TPE instead of naïve Gaussian Processes-based (GP BO) Bayesian Optimization was motivated by better dimensional scaling and internal support of both numerical and categorical data types (however, some minor transformations are still required). Unlike GP where optimization done by modeling distributions of Configuration results given it parameters, TPE builds two *parameter distributions* splitting Configurations into two sets according to their 'goodness'. For more detailed explanation we refer to original TPE description [4].

The other part of BOHB, namely HyperBand is a promising multi-armed bandit strategy for hyper-parameter optimization [38] 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 the maximum budget gives you the most precise Configuration evaluation while minimum amount results in the least accurate approximation of result. Running examples of budget could be the number of iterations in iterative algorithm or time to train the neuron network. As the result, requirements arise for TS to expose and support budget usage as expected in algorithm. At each iteration, HB randomly samples a number of Configurations for evaluation which, in fact, decreases for former 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. At each iteration of HB, the Successful Halving (SH) procedure is executed to drop (usually $\frac{2}{3}$) badly performing Configurations. As one could expect, since the number of measuring Configurations in each iteration decreases, the measurements could be done more precisely since the amount of SHs execution drops too.

Binding of HyperBand and Bayesian Tree Parzen Estimator exists 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 model built with results obtained with the largest budget only. This decision leads in more precise predictions in the later stages of parameter optimization procedure.

The drawback of this system lays in the way of handling conditions between hyperparameters. BOHB actual implementation HpBandSter (a distributed HyperBand implementation on Steroids) [3] as SMACv3 system uses ConfigSpace framework for Configuration Space definition. As we discussed in SMACv3 description (section 2.3.2), it naturally allows to encode the dependencies and conditions among parameters within space. As authors stated, the TPE learning models accomplish this knowledge implicitly by shrinking the densities for forbidden parameters (actually those parameters are still added to models by *imputation* mechanism where empty or default values are assigned). However, consider 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. The actual number of such parameters could vary dramatically in 'sparse' Search Spaces those distributions estimated by KDEs will not reflect the reality. As a consequence, the prediction performed using such distributions will often result in 'invalid' Configuration within 'sparse' spaces hurting the performance and accuracy of sampling.

[3] GitHub repository: `https://github.com/automl/HpBandSter`

AUTO-SKLEARN

Architecture search.

CASH (Combined Algorithm Selection and Hyperparameter optimization) problem

pros and cons (on-line or off-line, problems to solve, extensibility) [autosklearn:feurer2015efficient]

BRISEv2

approach description

Yevhenii: Other systems?

2.4 Parameter control as an Optimization Problem

2.4.1 Parameter Control Definition

2.4.2 Examples and Reported Impact

impact of parameter control based on other's evaluation

2.5 Conclusion

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 [surv:kerschke2019automated] page 6

3.3 Low Level Heuristics

Importance explanation

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 2.3) 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

BRISEv2

Yevhenii: Maybe, smth else..

4.1.3 Conclusion

BRISEv2 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 2. Multi Armed Bandit 3. Sklearn Bayesian Optimization 4. Static selection of SA, GA, ES	1. Default 2. Tuned beforehand 3. Random 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 this settings will be discussed in details in 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

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 [28]

Hyper-Heuristics with parameter tuning (or better say, control?), Constrainted 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 ⌣.

7 Future work

add more sophisticated models

dependencies / constraints in search space

add new class of problem (jmetalpy easly 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

Reviewer: consider merging with conclusion, if too short

Bibliography

[1] Satyajith Amaran et al. “Simulation optimization: a review of algorithms and applications”. In: *Annals of Operations Research* 240.1 (2016), pp. 351–380.

[2] David L Applegate et al. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[3] James Bergstra and Yoshua Bengio. “Random search for hyper-parameter optimization”. In: *Journal of machine learning research* 13.Feb (2012), pp. 281–305.

[4] James S Bergstra et al. “Algorithms for hyper-parameter optimization”. In: *Advances in neural information processing systems*. 2011, pp. 2546–2554.

[5] Leonora Bianchi et al. “A survey on metaheuristics for stochastic combinatorial optimization”. In: *Natural Computing* 8.2 (2009), pp. 239–287.

[6] Lorenz T Biegler and Ignacio E Grossmann. “Retrospective on optimization”. In: *Computers & Chemical Engineering* 28.8 (2004), pp. 1169–1192.

[7] Mauro Birattari et al. “Classification of Metaheuristics and Design of Experiments for the Analysis of Components Tech. Rep. AIDA-01-05”. In: (2001).

[8] 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.

[9] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. “A survey on optimization metaheuristics”. In: *Information Sciences* 237 (2013), pp. 82–117.

[10] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[11] Edmund Burke et al. “Hyper-heuristics: An emerging direction in modern search technology”. In: *Handbook of metaheuristics*. Springer, 2003, pp. 457–474.

[12] Edmund K Burke et al. “A classification of hyper-heuristic approaches: revisited”. In: *Handbook of Metaheuristics*. Springer, 2019, pp. 453–477.

[13] 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.

[14] 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.

[15] Wikimedia Commons. *File:Metaheuristics classification fr.svg — Wikimedia Commons the free media repository*. 2017.

[16] William Jay Conover and William Jay Conover. “Practical nonparametric statistics”. In: (1980).

[17] Juan De Vicente, Juan Lanchares, and Román Hermida. “Placement by thermodynamic simulated annealing”. In: *Physics Letters A* 317.5-6 (2003), pp. 415–423.

[18] Kalyanmoy Deb. “Multi-objective optimization”. In: *Search methodologies*. Springer, 2014, pp. 403–449.

[19] John H Drake et al. “Recent advances in selection hyper-heuristics”. In: *European Journal of Operational Research* (2019).

[20] 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.

[21] AE Eiben and JE Smith. “Popular Evolutionary Algorithm Variants”. In: *Introduction to Evolutionary Computing*. Springer, 2015, pp. 99–116.

[22] Agoston E Eiben and James E Smith. “What is an evolutionary algorithm?” In: *Introduction to Evolutionary Computing*. Springer, 2015, pp. 25–48.

[23] Stefan Falkner, Aaron Klein, and Frank Hutter. “BOHB: Robust and efficient hyperparameter optimization at scale”. In: *arXiv preprint arXiv:1807.01774* (2018).

[24] Goncalo Figueira and Bernardo Almada-Lobo. “Hybrid simulation–optimization methods: A taxonomy and discussion”. In: *Simulation Modelling Practice and Theory* 46 (Aug. 2014).

[25] Fedor V Fomin and Petteri Kaski. “Exact exponential algorithms”. In: *Communications of the ACM* 56.3 (2013), pp. 80–88.

[26] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 174. freeman San Francisco, 1979.

[27] Oded Goldreich. *P, NP, and NP-Completeness: The basics of computational complexity*. Cambridge University Press, 2010.

[28] Giovani 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.

[29] 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.

[30] 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.

[31] Frank Hutter et al. “ParamILS: an automatic algorithm configuration framework”. In: *Journal of Artificial Intelligence Research* 36 (2009), pp. 267–306.

[32] Lester Ingber. “Adaptive simulated annealing (ASA): Lessons learned”. In: *arXiv preprint cs/0001018* (2000).

[33] 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.

[34] Pascal Kerschke et al. “Automated algorithm selection: Survey and perspectives”. In: *Evolutionary computation* 27.1 (2019), pp. 3–45.

[35] Patrick Koch et al. “Automated hyperparameter tuning for effective machine learning”. In: *Proceedings of the SAS Global Forum 2017 Conference*. 2017.

[36] Pedro Larranaga et al. “Genetic algorithms for the travelling salesman problem: A review of representations and operators”. In: *Artificial Intelligence Review* 13.2 (1999), pp. 129–170.

[37] Niklas Lavesson and Paul Davidsson. "Quantifying the impact of learning algorithm parameter tuning". In: *AAAI*. Vol. 6. 2006, pp. 395–400.

[38] 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.

[39] Shen Lin and Brian W Kernighan. "An effective heuristic algorithm for the traveling-salesman problem". In: *Operations research* 21.2 (1973), pp. 498–516.

[40] M. Lindauer et al. "BOAH: A Tool Suite for Multi-Fidelity Bayesian Optimization & Analysis of Hyperparameters". In: *arXiv:1908.06756 [cs.LG]* (2019).

[41] Manuel López-Ibáñez et al. "The irace package: Iterated racing for automatic algorithm configuration". In: *Operations Research Perspectives* 3 (2016), pp. 43–58.

[42] 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.

[43] 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.

[44] Mustafa Mısır et al. "An intelligent hyper-heuristic framework for chesc 2011". In: *International Conference on Learning and Intelligent Optimization*. Springer. 2012, pp. 461–466.

[45] 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.

[46] 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.

[47] 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.

[48] Judea Pearl. "Intelligent search strategies for computer problem solving". In: *Addision Wesley* (1984).

[49] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[50] Nelishia Pillay and Derrick Beckedahl. "EvoHyp-a Java toolkit for evolutionary algorithm hyper-heuristics". In: *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2017, pp. 2706–2713.

[51] Patricia Ryser-Welch and Julian F Miller. "A review of hyper-heuristic frameworks". In: *Proceedings of the Evo20 Workshop, AISB*. Vol. 2014. 2014.

[52] Kenneth Sörensen, Marc Sevaux, and Fred Glover. "A History of Metaheuristics". In: *Handbook of Heuristics* to appear (Jan. 2017).

[53] 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.

[54] 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.

[55] 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.

[56] Enrique Urra Coloma et al. “hMod: A software framework for assembling highly detailed heuristics algorithms”. In: *Software Practice and Experience* 2019 (Mar. 2019), pp. 1–24.

[57] 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.

[58] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[59] Gerhard J Woeginger. “Exact algorithms for NP-hard problems: A survey”. In: *Combinatorial optimization—eureka, you shrink!* Springer, 2003, pp. 185–207.

[60] 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, 30th March 2020

Yevhenii Semendiak