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: 15th 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

List of Figures

List of Tables

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 [15] → exploration-exploitation balance
- no-free-lunch (NFL) theorem [24] → motivation of the thesis

thesis motivation The most related research field is Hyper-heuristics optimizations [5], 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 [15])*

Analysis of existing studies in field of parameter control and algorithm configuration problems *(find a problem definition)* [16]

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 2.2.4 and Systems for parameter tuning in 2.3.3 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 [14].

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, 4, 11] 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

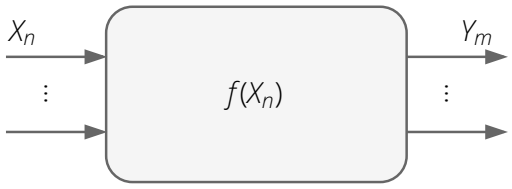

Figure 2.1 Target System

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, 4, 11] 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 [8].

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 be NP-complete computational complexity [13].

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 [7]). 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 [4].

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 [23], 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 [12], the objective of an exact algorithm is to perform better (in terms of running time) than exhaustive search. In both works [23] 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 [23] 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 [22]. 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 [22] 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 [22]:

- **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 [22].

A **greedy algorithm** is a quite different method that while constructing the solution, follows the logic of making a sequence of locally optimal decisions. Indeed, these decisions might not lead to a globally optimal solution, but the advantage of greedy algorithms is a simplicity in implementation [22].

Yevhenii: somewhere I found this alg in approximate solvers, otherwhere in heuristics..

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 [19]. 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.

Heuristic approaches could be discerned by the level of generality:

- **Simple heuristics** are algorithms, specifically designed to tackle concrete problem. They use the domain knowledge from Optimization Problem to gain a performance. For instance, while solving a Traveling Salesman Problem, heuristic approach could suggest proceeding with the next closest city from currently discovered. Simple heuristics do not provide any mechanisms to escape a local optimum and those could be easily trapped to it [19].
- **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 [3].
- **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 [5].

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

Motivation of Heuristics

An old engineering slogan says, "Fast, Cheap or Good? Choose two." And here we should make a decision now, 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.

In third hand (why not?) 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.

Yevhenii: will continue...

2.2 Heuristic Solvers for Optimization Problems

TSP as the running example. I guess, I will introduce it as an example of perturbation problems in previous section.??

2.2.1 Heuristics

Definition

Examples

2.2.2 Meta-Heuristics

Definition

Classification

Examples

We distinguish following examples among all existing meta-heuristics, since later we use them as the LLH in developed hyper-heuristic.

GA

SA

ES

2.2.3 Hybrid-Heuristics

Definition

Examples

Guided Loca Search (GLS) + Fast Local Search [21]

Direct Global + Local search [20]

Simulated Annealing + Local Search [18]

No-Free-Lunch Theorem

NFL is the problem of heuristics[24]

Exploration-Exploitation Balance

Conclusion

Proper assignment of hyper-parameters has great impact on exploration-exploitation balance and those on (meta) -heuristic performance.

2.2.4 Hyper-Heuristics

Definition

Classification

Search Space: heuristic selection, heuristic generation

Learning time: on-line learning hyper-heuristics, off-line learning hyper-heuristics, no-learning hyper-heuristics

Other classification characteristics from [15], [6], mb smth else. For instance, hyperparameter tuning

Examples

[9] (Online algorithm selection at page 27); [15]

2.2.5 Conclusion on Approximate Solvers

Pros and cons of heuristics - Heuristics are strictly problem dependent and each time require adaptations.

Pros and cons of meta-heuristics - no LLH selection, strict to one problem

Pros and cons of hybrid-heuristics - no LLH selection, strict to one problem ?

Pros and cons of hyper-heuristics - no parameter control?

2.3 Parameter Tuning as a Search Problem

The goal of section: analysis of existing systems for hyper-parameter optimization (tuning), weaknesses and strength of each of the system

2.3.1 Parameter Tuning Problem Definition

2.3.2 Approaches for Parameter Tuning

Grid Search

Random Search

Model Based Search

2.3.3 Systems for Model Based Parameter Tuning

IRACE

approach [17]

pros and cons

SMAC

approach description

BOHB

approach description

AUTO-SKLEARN

CASH (Combined Algorithm Selection and Hyperparameter optimization) problem

pros and cons (on-line or off-line, problems to solve, extensibility) [10]

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

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 [15] 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	1. Default
2. Multi Armed Bandit	2. Tuned beforehand
3. Sklearn Bayesian Optimization	3. Random
4. Static selection of SA, GA, ES	4. Tree Parzen Estimator
	5. Sklearn Bayesian Optimization

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

For our benchmark we selected following settings sets:

- *Baseline:* 4.1, 4.2;

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

Each of 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

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

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

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

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

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

[7] 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.

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

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

[10] Matthias Feurer et al. “Efficient and robust automated machine learning”. In: *Advances in neural information processing systems*. 2015, pp. 2962–2970.

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

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

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

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

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

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

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

[18] 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.

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

[20] 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.

[21] 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.

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

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

[24] 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, 15th March 2020

Yevhenii Semendiak