Assignment of master's thesis

Title: Construction of hyper-heuristics for non-deterministic polynomial problems in SEAGE

Student: Bc. David Omrai
Supervisor: Ing. Mgr. Ladislava Smítková Janků, Ph.D.
Study program: Informatics
Branch / specialization: Knowledge Engineering
Department: Department of Applied Mathematics
Validity: until the end of summer semester 2023/2024

Instructions

Build on the results of your previous work connecting together the HyFlex hyper-heuristics evaluator and the SEAGE optimization framework. In your thesis, focus on construction at least two hyper-heuristics for non-deterministic polynomial problems implemented in the SEAGE framework such that at least one is comparable or better than the current known hyper-heuristics, and at least one is constructed to test a hypothesis regarding the use of hyper-heuristics that prefer simple blocks (heuristics) or complex blocks (meta-heuristics).

1. Conduct a search of the current state of research in the field.
2. Design and implement an advanced hyperheuristic that achieves comparable or better results than the hyperheuristics from CHeSC2011.
3. Implement the best hyperheuristics from CHeSC2011 in the SEAGE framework using the implemented metaheuristics.
4. Design and implement experiments.
5. Evaluate the results, confirm the verification or rejection of the hypothesis.


CONSTRUCTION OF HYPER-HEURISTICS FOR NON-DETERMINISTIC POLYNOMIAL PROBLEMS IN SEAGE

Bc. David Omrai

Faculty of Information Technology
Department of Applied Mathematics
February 13, 2024
Contents

Acknowledgments vii
Declaration viii
Abstract ix
List of Abbreviations x

1 Introduction 1
1.1 Motivation .......................................................... 1
1.2 Aim of this Work .................................................... 1
1.3 Current State of Hyper-heuristics Study ...................... 2
1.4 Structure of the Work ............................................. 3

2 Theory 5
2.1 Optimization Problems ........................................... 5
  2.1.1 SAT .................................................................. 6
  2.1.2 TSP .................................................................. 7
  2.1.3 JSP .................................................................. 7
  2.1.4 FSP .................................................................. 8
  2.1.5 Rosenbrock Problem ........................................... 8
2.2 Optimization .......................................................... 9
2.3 Deterministic Algorithms .......................................... 9
2.4 Heuristics ............................................................. 9
  2.4.1 Random Search ................................................ 9
  2.4.2 Greedy Algorithm ............................................. 10
2.5 Metaheuristics ....................................................... 10
  2.5.1 Metaheuristics Classification .............................. 10
  2.5.2 Genetic Algorithm ............................................ 11
  2.5.3 Tabu Search ..................................................... 12
  2.5.4 Simulated Annealing ......................................... 13
  2.5.5 Ant Colony ....................................................... 14
2.6 Hyper-heuristics ..................................................... 15
  2.6.1 Hyper-heuristics Classification ............................ 15
2.7 Optimization Frameworks ......................................... 16
2.8 Metric ................................................................. 16

3 Unit Metric 17
3.1 Definition ............................................................ 17
3.2 Quality of the Results ............................................. 17
3.3 Metadata ............................................................ 17
3.4 Instance Equation .................................................. 18
3.5 Problem Equation ................................................ 19
3.6 Experiment Equation ............................................. 19
List of Figures

2.1 The schedule of 4x3 (problems x machines) job shop scheduling problem with total time-span $C_{max} = 28$. ................................................................. 7
2.2 The schedule of 4x3 (problems x machines) flow shop scheduling problem with total time-span $C_{max} = 29$. ................................................................. 8
2.3 Parts of the Genetic Algorithm. ................................................................. 11
2.4 Tabu Search in the maze (after finding the optimum 0, it continues with the search). 12
2.5 Convergence of the Simulated Annealing[36]. ............................................. 13
2.6 Ant Colony Optimization process. .............................................................. 14
2.7 A classification of hyper-heuristic approaches according to two dimensions (feedback, nature of heuristic search space).[9] ....................................................... 15
3.1 The Unit metric (score) of an imaginary algorithm’s solution for a certain problem instance. ................................................................. 18
3.2 Heatmap color range. ................................................................. 20
4.1 Diagram of HyFlex with implemented evaluator. ........................................... 22
5.1 SEAGE’s architecture.[40] ................................................................. 25
5.2 Diagram of SEAGE with implemented evaluator inside. ................................ 27
6.1 LeanGIHH flowchart. ................................................................. 33
6.2 VNS-TW flowchart. ................................................................. 36
6.3 ISEA flowchart. ................................................................. 39
6.4 DOHH flowchart. ................................................................. 44
6.5 DOHH experiment metaheuristics schedule on TSP instance u2152 [60 s, 1 repeat]. 45

List of Tables

3.1 Problem instances heatmap table preview. ................................................... 20
3.2 Problem domains heatmap table preview. .................................................. 20
4.1 Heatmap leaderboard of hyper-heuristics in HyFlex run 120s, 5 repeats. ........ 24
5.1 SEAGE’s implementation matrix.[40] ......................................................... 26
5.2 Score heatmap of SEAGE metaheuristics run 120s, 5 repeats. ....................... 28
7.1 Instances of each problem domain used in the experiments. ......................... 46
7.2 HyFlex hyper-heuristics experiment scores 60 seconds, 4 repeats. ................. 48
List of code listings

3.1 Metadata of the TSP problem domain instance .......................... 18
3.2 Metadata of the SAT problem domain instance .......................... 18
3.3 Metadata of the FSP problem domain instance .......................... 18
3.4 Metadata of the JSP problem domain instance .......................... 18
3.5 Preview of algorithm’s scorecard ........................................ 19
4.1 Example command of HyFlex experiment run ............................ 23
4.2 Example command of HyFlex evaluation .................................. 23
5.1 Example command of SEAGE metaheuristics experiment run ............ 27
5.2 Example command of SEAGE hyper-heuristic experiment run .......... 28
5.3 Example command of SEAGE report experiments ........................ 28
I would like to express my sincere gratitude to my supervisor, Ing. Mgr. Ladislava Smítková Jánká, Ph.D., for her patient and professional guidance throughout the completion of this master’s thesis. Special thanks to Ing. Richard Málek, the author of the SEAGE framework, for his assistance in its development. Lastly, I extend gratitude to my family and friends for their unwavering support. A special thanks to my best friend and family member, Aimee, for being always there throughout my studies.
Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons who wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location, and quantity. However, all persons that make use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works, or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on February 13, 2024
Abstract

The primary focus of this thesis is to build upon the results of my previous work (bachelor’s thesis), where I implemented a method to evaluate and compare the results of optimization algorithms, including hyper-heuristics, and metaheuristics, in an understandable way. The evaluated result is represented by a unified numerical value indicating how close or far it is from the optimum. This was done in two optimization frameworks: SEAGE and HyFlex, using a newly designed metric called the Unit metric.

In this work, I have expanded the usability of the SEAGE framework to four problem domains. Additionally, I have improved the metaheuristics, introduced new methods for accessing and utilizing the database to manage data from prior experiments, and implemented a new visualization method for a better comparison of experiment results.

With this expanded environment, I have implemented a new hyper-heuristic and re-implemented three top-performing hyper-heuristics from the CHeSC2011 competition (originally implemented in the HyFlex framework) within the SEAGE framework. To address the performance of my newly implemented hyper-heuristic and the hypothesis of this work concerning how performance differs when hyper-heuristics utilize metaheuristics (in the SEAGE) instead of low-level heuristics (in the HyFlex), I’ve designed a set of experiments. These experiments revealed that employing low-level heuristics achieved slightly better results than using metaheuristics on most problem domains, and my hyper-heuristics proved to be on par with CHeSC2011 hyper-heuristics.

Keywords
Hyper-heuristic, Metaheuristic, Heuristics, SAT, TSP, JSP, FSP, Thompson’s sampling, Evaluator, LeanGIHH, AdapHH, GIHH, VNS-TW, ISEA, DOHH, SEAGE, CHeSC2011, HyFlex

Abstrakt

Hlavním cílem této diplomové práce je navážat na výsledky mé bakalářské práce, ve které jsem vytvořil metodu pro srozumitelnější ohodnocení a srovnání řešení optimalizačních algoritmů, včetně hyper-heuristik a metaheuristik. Vyhodnocené řešení je reprezentováno sjednocenou numerickou hodnotou, která ukazuje, jak blízko nebo daleko je od optimálního řešení. Toho jsem realizoval ve frameworku SEAGE i HyFlex s využitím nově navržené metriky nazvané Unit metrika.

V rámci této práce jsem rozšířil použitelnost frameworku SEAGE na celkem čtyři domény problémů, naimplementoval nové způsoby přístupu k databázi SEAGE pro manipulaci s daty z předchozích experimentů, vylepšil metaheuristiky a představil novou vizualizaci kvality řešení experimentů pro jejich srozumitelnější porovnání.

S tímto rozšířeným prostředkem jsem v rámci frameworku SEAGE naimplementoval novou hyper-heuristiku a přeimplementoval trojici nejlepších hyper-heuristik z výzvy CHeSC2011, které byly původně implementovány ve frameworku HyFlex. Pro sledování výkonu mé nově implementované hyper-heuristiky a rozhodnutí o hypotéze této práce, týkající se rozlišitelnosti výkonu hyper-heuristik využívaných metaheuristiky (v SEAGE) místní heuristik (v HyFlex), jsem navrhl několik experimentů. Tyto experimenty ukázaly, že ve většině domén optimalizačních problémů dosahuje využití jednoduchých heuristik mírně lepších řešení než použití metaheuristik. Dále experimenty ukázaly, že má hyper-heuristika je srovnatelná s hyper-heuristikami z výzvy CHeSC2011.

Klíčová slova

Hyper-heuristika, Metaheuristika, Heuristika, SAT, TSP, JSP, FSP, Thompsonovo vzorkování, Evaluator, LeanGIHH, AdapHH, GIHH, VNS-TW, ISEA, DOHH, SEAGE, CHeSC2011, HyFlex
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>SAT</td>
<td>Boolean Satisfiability Problem</td>
</tr>
<tr>
<td>TSP</td>
<td>Traveling Salesman Problem</td>
</tr>
<tr>
<td>FSP</td>
<td>Flow Shop Problem</td>
</tr>
<tr>
<td>JSP</td>
<td>Job Shop Problem</td>
</tr>
<tr>
<td>ACO</td>
<td>Ant Colony Optimization</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>SA</td>
<td>Simulated Annealing</td>
</tr>
<tr>
<td>TS</td>
<td>Tabu Search</td>
</tr>
<tr>
<td>AdapHH</td>
<td>Adaptive Hyper-Heuristic</td>
</tr>
<tr>
<td>VNS-TW</td>
<td>Variable Neighborhood Search Taiwan Hyper-Heuristic</td>
</tr>
<tr>
<td>ISEA</td>
<td>Iterated Local Search Hyper-Heuristic</td>
</tr>
<tr>
<td>DOHH</td>
<td>David Omrai Hyper-Heuristic</td>
</tr>
<tr>
<td>CHeSC2011</td>
<td>Cross-Domain Heuristic Search Challenge 2011</td>
</tr>
<tr>
<td>HyFlex</td>
<td>Hyper-Heuristics Flexible Framework</td>
</tr>
<tr>
<td>SEAGE</td>
<td>Search Agents Framework</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

The study of optimization problems and how to solve them through algorithms and heuristics can be traced back to the start of civilization. However, when we narrow our focus to the well-documented heuristic study, its beginnings can be dated to the first four decades of the 20th century. [1]

For a long time, the study of heuristics wasn’t taken seriously. The opinion was that, compared with algorithms, it lacked a certain analytical clarity earned through academic study. Nevertheless, it’s hard to argue against the contribution and usefulness of solving real-world problems. [2]

Heuristics can be sorted into three main categories: low-level heuristics, metaheuristics, and hyper-heuristics (the focus of this work). [3] Low-level heuristics are adept at solving problems too complex for classical algorithmic methods but, for instance, lack the self-awareness to not get stuck in local optima. Metaheuristics attempt to solve these problems, but their effectiveness also depends on the values of their parameters. The performance of metaheuristics depends on proper parameter settings, which can be challenging to obtain. Hyper-heuristics attempt to address this issue. Certain hyper-heuristics aim to accompany the low-level methods in the search process to yield the best possible solution. They set parameters, create plans with multiple different (meta)heuristics, and make other intelligent decisions to maximize the potential of the used parts.

The hidden potential of hyper-heuristics is the main motivation behind this work, aiming to showcase the implementation and comparison of sophisticated methods in a strong, reliable, and easy-to-use framework, SEAGE.

1.2 Aim of this Work

In the field of (meta/hyper)heuristics, numerous works showcase innovative approaches, claiming to demonstrate promising and effective performance ([4], [5], [6], [7]). Whether by adapting processes observable in nature or employing other intelligent techniques for solving computational problems, the problem with these claims is that there is no source code, and not always they can be reproduced and compared with others in an unbiased environment.

The response to this deficiency is addressed by the HyFlex framework, specially introduced for the CHeSC2011 competition, where anyone could participate with their hyper-heuristics. It provides a simple API for utilizing low-level heuristics, enabling the easy creation of hyper-heuristics. This attracted a wide range of authors with their unique ideas. The downside of this
competition and the framework itself was that there was no metric to address and explain the independent strengths of a single hyper-heuristic. There was only the F1 metric (Section 4.1), which sorts the hyper-heuristics based on their performance but does not provide an independent value for a single hyper-heuristic. The resulting value can be interpreted only in one evaluation of the set of hyper-heuristics results. Questions, ‘What problem instances is my hyper-heuristic weak/strong on, and to what extent?’ remained unanswered.

Yet, the implemented framework with all these HyFlex (well-documented) hyper-heuristics offers a key thing: reproducible results. This was the aim of my previous work: in my bachelor’s thesis, and during the VýLet 2021-2022 CTU FIT study program, I’ve introduced a new metric called the Unit metric (more on that in the Chapter 3) that assesses the quality of each result and maps it to the (0, 1) interval. The closer it is to 1, the closer it is to the optimal value; the closer it is to 0, the closer it is to an easily accessible result (greedy algorithm). This metric can be applied to the problem instance result, the problem domain results, or the entire experiment results involving multiple problem domains. HyFlex was particularly made to offer easy development of (selection) hyper-heuristics, but what if I want to compare hyper-heuristics with metaheuristics or simple heuristics? And what if I want to understand closely each algorithm through gathered data from previous experiments?

This is where the SEAGE framework comes in handy with its library of optimization problems and the implementation of algorithms. Anyone can easily develop their algorithm, heuristic, metaheuristic, or now even hyper-heuristics, and compare the results with HyFlex hyper-heuristics using the same metric.

This work aims to develop a part of the SEAGE framework to allow straightforward implementation of hyper-heuristics. By re-implementing a few HyFlex hyper-heuristics in SEAGE, I am testing how their behavior changes by using metaheuristics instead of low-level heuristics, and finally implementing my own to showcase the simple implementation itself, and challenging the other hyper-heuristics.

1.3 Current State of Hyper-heuristics Study

Hyper-heuristic is a relatively new term. Its definition was coined around 2000 [9] and is understood as ‘heuristic selecting heuristics.’ However, the idea of high-level heuristics can be traced back to the first half of the 1960s. [10]

In the last few years, several works, studies, and articles have been done. I will mention a few important ones.

In 2011, an international challenge, CHeSC2011, took place. [11] During this event, participants had the opportunity to compare their hyper-heuristics with each other. Hyper-heuristics were developed in the HyFlex framework, which was implemented for this event. [12] One of the main advantages of this framework is the representation of low-level heuristics. Thanks to this, participants can solely focus on constructing the best hyper-heuristics with no additional problems. From this challenge came several works that changed the course of subsequent research in this field. One of them is LeanGiHH [13], a heuristic built upon the AdapHH [14] winning hyper-heuristic of this challenge. It employs the previous idea but tries to simplify the concept and successfully reduces the previous 2,324 lines of code to just 288.

In 2013, an article titled Hyper-heuristics: a survey of the state of the art [15] appeared, summarizing the entire academic study up to 2012. It covers the history, trends, and the path that the hyper-heuristics field will continue to follow. The authors introduced new categories of hyper-heuristics, with two main ones called selective and generative. Selective hyper-heuristics use already constructed low-level heuristics for their tasks, while generative hyper-heuristics produce new low-level heuristics. The main goal of the authors was to encourage groups orbiting around similar fields (metaheuristics and machine learning) to work together.

In 2016, a new design featuring three useful methods for generating heuristics by hyper-
heuristics is described in the article *Automated design of production scheduling heuristics: A Review* [16]. These concepts include a new representation of candidate heuristics defining the search space, optimization algorithms for the search space, and a fitness function used for the quality of candidate heuristics. The authors also categorize hyper-heuristics into supervised and unsupervised [16].

One of the latest articles is from 2019, titled *Recent advances in selection hyper-heuristic* [10]. The primary focus of this article is selective hyper-heuristics and the introduction of critical discussions on the new trends and the path this field is taking. The article also introduces the HyFlex framework and the international challenge CHeSC2011. It describes how this event was organized, how each of the hyper-heuristics earned points, and how new works are affected by this framework.

### 1.4 Structure of the Work

This work continues where my bachelor’s thesis [8] left off, with a newly implemented metric (called the Unit metric, as mentioned earlier—a way to evaluate results in an easy-to-understand way, detailed in Chapter 3) in both the frameworks (HyFlex and SEAGE) and a basic capability to compare metaheuristics and hyper-heuristics with each other.

The goals of this work include deeper integration of using the Unit metric and results visualization methods (heatmap table) in SEAGE, implementing a new hyper-heuristic to achieve results comparable or superior to those from the CHeSC2011 competition, re-implementing the best hyper-heuristics from that competition in the SEAGE framework, and addressing the hypothesis concerning different implementation approaches, specifically, whether to utilize heuristics or metaheuristics. To address the challenge of gathering enough data to either accept or reject the hypothesis, I’ve re-implemented three top-performing hyper-heuristics from the CHeSC2011 competition, implemented in the HyFlex framework, (covered in Chapter 6). I’ve designed a set of experiments to evaluate how performance differs when hyper-heuristics utilize metaheuristics instead of heuristics (covered in Chapter 7).

For the implementation of a new hyper-heuristic, I’ve chosen the SEAGE framework to showcase the development of new hyper-heuristics in this expanded environment, detailed in Chapter 6. Finally, to evaluate the performance of this new hyper-heuristic compared to those from CHeSC2011, I’ve designed a series of experiments. These experiments involve comparing the new hyper-heuristic with the original implementation, then with the re-implemented variation, and finally with the individual blocks, it utilizes (metaheuristics) (covered in Chapter 7). This approach helps determine whether the use of more complex blocks is worthwhile.

But the possible progress in developing hyper-heuristics in the SEAGE, and answering the hypothesis of this work had a long way to go. There were only two problem domains on which the algorithms could run, there were no easily understandable visualizations of experiments, and most of the metaheuristics needed to be worked on to enhance their performance.

All these mentioned problems were also the subject of my work on this thesis. I’ve successfully extended the number of problem domains (to four), three of which are also available in the HyFlex framework, helping this work for a wider and more comprehensive comparison of algorithms implemented in both frameworks.

To prepare the capability for hyper-heuristics to store and access previously used metaheuristic parameters, enhancing greatly their performance, I’ve implemented a component that stores information from prior experiments, including the parameters used in metaheuristics and additional details about the quality of the found problem solutions, duration, and other relevant information from previous instances. Further insights into this implementation can be found in Chapter 6.

In addition to parameters, the hyper-heuristics require a set of solutions, that they handle during the search process. Before my work, the only possible implementation of this was to
implement it by each hyper-heuristic. This challenge has been addressed with the introduction of a new way to manage solutions, allowing hyper-heuristics to easily access the solutions—more on that in Chapter 6.

Before I could proceed with the implementation of hyper-heuristics in SEAGE and address the hypothesis of this work (comparing hyper-heuristics utilizing heuristics with hyper-heuristics utilizing metaheuristics), there was an issue with understanding the quality of the experiment results. While the used Unit metric aims for simplicity, an increasing number of evaluated results in the result table (performance on each problem domain and instance) made it progressively challenging to quickly discern the performance of each algorithm.

The interpretation of results stands as a cornerstone of this work and demands simplicity. To address this, I’ve implemented a visualization method that utilizes colors to distinguish between the best, good, and poor results, greatly enhancing the simplicity of results interpretation. This has been implemented in both frameworks to help visualize the experiments in this work. More on that in Chapter 3.

The findings and observed performance are summarized and analyzed in Chapter 8 providing insights into potential future directions within this field.
Chapter 2
Theory

2.1 Optimization Problems

There are various interpretations of optimization problems in this category. They can be linear or nonlinear, convex or non-convex, constrained or non-constrained. However, the primary objective of all these problems remains the same—to obtain the best solution, also known as optimal. The process of searching for the optimum is called optimization, as we aim to optimize the starting solution, transforming it (hopefully) into the optimal solution for the given problem. The approach that numerous optimization algorithms and strategies adopt varies depending on how the problem is defined and the solution space in which it operates.[17]

Discrete The variables that are being optimized are from a discrete interval. (SAT for instance)

Continuous The variables that are being optimized are from a continuous interval. (Rosenbrock problem for instance)

As I mentioned, the solutions are being optimized, but there has to be some value to be optimized. For this, the objective function is used. Solution parameters that can be changed are called control or decision variables, and restrictions on allowed parameter values are known as the already-mentioned constraints. There are several types of constraints, primarily equality constraints, and inequality constraints. The set of candidate solutions that satisfy all constraints is then called the feasible set.

Definition 2.1. (Optimization Problem)

\[
\begin{align*}
\text{function } & f: \mathbf{A} \mapsto \mathbb{R} \\
\text{optimize } & \text{argmin}_x f(x) \\
\text{subject to } & c_i(x) \leq 0 \quad i = 1, \ldots, m, \: m \in \mathbb{N}_0 \\
& c_j(x) = 0 \quad j = 1, \ldots, p, \: p \in \mathbb{N}_0
\end{align*}
\]

Here the \( f(x) \) is the objective function, \( x \) is the optimization variable of the problem, the \( c_i(x) \) is the inequality constrain function, and finally \( c_j(x) \) is the equality constraint. [17] [18] For example imagine a simple optimization problem \( \min f(x) = x_1^4 + x_2^2 \), where \( x \) denotes the vector \((x_1, x_2)\), and two constraints \( x_1 \leq 9 \) and \( x_2 = 6 \). In this presented example, the objective function \( f(x) \) defines what is to be minimized (objective value), and two hard constraints define feasible solutions. Without constraints, the solution would be \( x = (0, 0) \), for this \( x \) has the lowest
objective value for this problem. The real solution satisfying both solutions is \( x = (9, 6) \), with objective value of 6597.

Before advancing any further I would like to recap a few important (even though simple) definitions of the properties of the objective function.

▶ **Definition 2.2. (Local optimum).** This refers to either local maximum or local minimum.

Local maximum \( (\exists \bar{x} \in A)(\exists \epsilon > 0)(\forall x \in A)(|\bar{x} - x| < \epsilon \Rightarrow f(\bar{x}) \geq f(x)) \quad A \subseteq \mathbb{R}^n \)

Local minimum \( (\exists \bar{x} \in A)(\exists \epsilon > 0)(\forall x \in A)(|\bar{x} - x| < \epsilon \Rightarrow f(\bar{x}) \leq f(x)) \quad A \subseteq \mathbb{R}^n \)

▶ **Definition 2.3. (Global optimum).** This refers to either the global maximum or the global minimum.

Global maximum \( (\exists \bar{x} \in A)(\forall x \in A)(f(\bar{x}) \geq f(x)) \quad A \subseteq \mathbb{R}^n \)

Global minimum \( (\exists \bar{x} \in A)(\forall x \in A)(f(\bar{x}) \leq f(x)) \quad A \subseteq \mathbb{R}^n \)

2.1.1 SAT

The Boolean satisfiability problem (SAT) is a task in mathematical logic that seeks to determine whether the variables of a given Boolean expression can be replaced by truth values or false values in a way that the resulting expression (formula) is satisfied. If we cannot achieve this goal and, for all possible assignments, the expression has the value false, then we consider the formula unsatisfiable. [19]

The Boolean satisfiability problem (SAT) is a decision problem concerning the satisfiability of a formula. There are several variations of the SAT Problem, and the specific variant employed in this work is MaxSAT; MaxSAT is an optimization extension of the SAT problem. Instead of determining if a Boolean formula can be satisfied, our objective is to find an assignment that *maximizes the number of satisfied clauses within the formula*. Typically formulated in *conjunctive normal form* (CNF), the Boolean formula consists of the conjunction of multiple clauses, where each clause is a disjunction of literals. Each clause often has a fixed number of literals, and when this number is specified, the problem is referred to as n-MaxSAT, where n is the designated number of literals in each clause. [20]

▶ **Definition 2.4. (Boolean Formula).** A logical expression constructed using Boolean operators (AND, OR, NOT) on Boolean variables. For 3-MaxSAT with 6 variables, one of the formulas can look as follows.

\[
\text{formula} =: (x_1 \lor x_2 \lor x_3) \land (x_2 \lor x_4 \lor x_5) \land (x_1 \lor x_3 \lor x_6)
\]

**Complexity** MaxSAT is an *NP-hard* problem, indicating that it is as challenging as the hardest problems in NP (nondeterministic polynomial time). Given its NP-hard nature, finding exact solutions can be computationally intensive; therefore, approximations are often used in practical applications.

**Applications** There is a vast array of applications, including AI planning, the optimization of scheduling problems, and the determination of optimal resource assignments.
2.1.2 TSP

The Traveling Salesman Problem (TSP) is a discrete optimization problem in which the goal is to find the shortest and most efficient route for a person across a set of towns, ensuring that each town is visited exactly once. \(^{21}\)

The problem is represented as a weighted graph, where nodes are cities, and edges between cities are assigned distances representing the travel cost. The objective is to find a permutation of cities (a tour), specifying the order in which the salesman visits the towns, that minimizes the total distance of visiting each city exactly once and returning to the starting city. \(^{22}\)

**Complexity** TSP is also known to be NP-hard, meaning with more cities, the problem becomes more computationally harder to get exact solutions.

**Applications** There is a vast array of applications, including logistics and transportation, manufacturing and scheduling, and networking (optimizing data packet routing in computer networks).

2.1.3 JSP

The Job Shop Scheduling problem (JSP) is a discrete combinatorial optimization problem comprising a set of different machines that perform operations on jobs. \(^{23}\)

Each job has a specified sequence of processing orders through the machines, represented by an ordered list of operations. Each operation is determined by the required machine and its associated processing time. Several constraints exist for both jobs and machines, including the order of jobs at each station and interruption policies. Each job can be processed only on one machine at a time, and each part of the job can be executed only once on each machine after the previous one has been processed. The goal is to determine the order in which the operations are processed on each of the machines to optimize a certain objective, typically minimizing the make-span. \(^{24}\)

**Complexity** JSP is also among the NP-hard, meaning that as the number of jobs and machines grows, finding optimal solutions becomes computationally challenging.

**Applications** We can see this problem in many applications of the real world. For instance manufacturing and production, and environments, where different jobs require different sequences of operations on various machines.

![Figure 2.1](attachment:image.png) The schedule of 4x3 (problems x machines) job shop scheduling problem with total time-span \(C_{\text{max}} = 28\).
2.1.4 FSP

The Flexible Job Shop Scheduling Problem (FSP) is a discrete combinatorial optimization problem, a special case of the Job Shop Problem. Describing FSP involves a set of machines and jobs. Each job consists of operations distributed across different machines. All jobs share the same sequence of processing operations when navigating through the machines. No precedence constraints exist among operations of different jobs. Operations do not allow interruptions, and each machine is limited to processing a single operation at a time. The objective is to determine the operation sequences for jobs (performed in the same order on all machines) that minimize the make-span.\[25\]

**Complexity** FSP is NP-hard, indicating computational complexity in finding optimal solutions.

**Applications** This problem can be found in various industries, where it plays a crucial part in optimization. For instance in manufacturing industries, to make efficient scheduling of operations on flexible production lines. In semiconductor manufacturing, for fabrication, the production of different types of chips involves a complex set of operations on machines with various capabilities.

![Figure 2.2](image-url)  
*Figure 2.2* The schedule of 4x3 (problems x machines) flow shop scheduling problem with total time-span $C_{\text{max}} = 29$.

2.1.5 Rosenbrock Problem

I’ve described all discrete optimization problems that are used in this work for the comparison of various strategies (metaheuristics and hyper-heuristics), but it wouldn’t be right to leave out the previously mentioned Rosenbrock problem, which is implemented in the SEAGE framework to test and benchmark strategies for solving continuous optimization problems.

The Rosenbrock problem refers to an optimization problem that involves the *Rosenbrock function* as an objective function. This problem is also known as the *Rosenbrock’s Valley* or *banana function* (the name is derived from how the plot of the Rosenbrock function of two variables looks).\[26\]

The multidimensional generalization of the Rosenbrock function is as follows.
Definition 2.5.

\[ f(x) = \sum_{i=1}^{N-1} [100(x_{i+1} - x_i)^2 + (1 - x_i)^2] \quad x \in \mathbb{R}^N \]

The optimization process can be efficiently performed by adapting an appropriate coordinate system without using any gradient information (without applying gradient descent) and without building local approximation models. A regular pattern that many of the stationary points show when plotted can be exploited to locate them. Optimization is usually achieved by adaptive coordinate descent from some starting point or the Nelder-Mead method (a numerical method to find the optimum in multidimensional space).

2.2 Optimization

Optimization is the process of finding the best possible (or defined quality) solution for a given problem. As I already mentioned, it involves maximizing or minimizing an objective function, subjected to a set of constraints or conditions. Optimization can be classified in many ways. For instance, it can be global or local optimization, meaning it aims to find the best solution or just the best within some subset of all possible solutions to achieve the objective. Regarding restrictions, there is also constrained and unconstrained optimization, where the model variables are subjected to explicit constraints or not. Lastly, optimization can be deterministic or non-deterministic, indicating whether the used model has determined a set of actions to achieve the solution or not.

2.3 Deterministic Algorithms

An algorithm is a description of an automated solution that operates in a precisely defined way to solve a problem. The key characteristic of deterministic algorithms is the absence of randomness, ensuring entirely reproducible and predictable execution (with the same input producing the exact output). Examples of such algorithms include bubble sort, merge sort, and finding the greatest common divisor.

2.4 Heuristics

Heuristics represent a problem-solving approach often employed when the optimal solution or algorithmic method becomes overly complex (unable to reach the feasible solution in a reasonable time). Heuristics consider these problems and aim to provide a good-enough (not guaranteed) solution effectively, in an easy-to-understand manner (for instance, a greedy algorithm takes the best option in each step). They prioritize practicality, seeking solutions that are good enough for the given purpose.

In practical scenarios, the pursuit of an optimal solution is not always necessary; a sufficiently good solution suffices. Consequently, heuristics do not solely focus on attaining a single best solution but explore a range of solutions with varying quality along the way. Their adaptability to diverse situations renders them flexible and applicable to a variety of problems.

2.4.1 Random Search

The simplest heuristic (optimization algorithm) is the random search, and its effectiveness lies in the random exploration of the solution space. Instead of following a deterministic or systematic
pattern, a random search takes each step as a guess. The major advantages of this exploratory strategy include easy implementation for parallel runs (covering a larger space), the ability to escape local extremes, and surprisingly effective performance in poorly understood solution spaces.\[30\]

Random search is commonly employed as a baseline exploration algorithm, particularly for initial exploration phases. Although this approach doesn’t guarantee finding the optimal solution, it still offers several advantages that make it a valuable tool for more complex optimization algorithms (metaheuristics and hyper-heuristics).

### 2.4.2 Greedy Algorithm

The greedy algorithm is a simple yet often efficient optimization strategy that makes local optima choices in each step it takes. The idea is straightforward, when I take the best possible step every time I should eventually get to the global optima. This would be the case for convex spaces, but that’s not always the case. The lack of consideration of the global context in more complex spaces results in falling into local optima from which it can’t escape (reaching the sub-optimal solution). \[31\]

Despite its cons, it’s widely used by many algorithms (Dijkstra’s Algorithm, Minimum Spanning Tree) and more complex optimization algorithms (metaheuristics and hyper-heuristics).

### 2.5 Metaheuristics

The prefix ‘meta’ originates from Greece and denotes something with a more general approach, addressing various fundamental heuristic problems. One such problem involves getting stuck in local optima, making it impossible to continue the search (optimization) process. Metaheuristics fine-tune and leverage the collaboration of lower-level heuristics, yielding significantly improved results compared to what any individual lower-level heuristic can achieve on its own.

Among the main features of metaheuristics is the capability to employ the same higher-level framework for various optimization problems. They maintain a balance between exploration and exploitation, a key strategy to avoid getting stuck in local extremes and to reach a larger solution space for obtaining near-optimal results. The downside of these strategies still exists, as they do not guarantee the discovery of high-quality solutions within a reasonable amount of time, and their performance is dependent on parameter settings, which can be challenging. Nonetheless, their flexibility to adapt to various problems through parametrization makes them a robust tool.\[32\]

#### 2.5.1 Metaheuristics Classification

Searching strategies can be classified by what search space they cover in their search strategies, and how many solutions they work with.

**Local Search** The simplest searching strategy of this type is already mentioned in greedy search. It tries to find better solutions in small solution space areas, not caring to find the global optimum (it’s mostly used in need of exploitation).

**Global Search** Exploitation by itself wouldn’t always achieve finding the best possible solutions for its incapability to get from local optima without help. That’s where the global search shines. It explores much larger solution space and picks samples of many promising areas. These areas can be then exploited to find the best possible solutions.

**Monte Carlo algorithms** Algorithms of this broad class use repeated random sampling to solve difficult or vast solutions space problems. A few of the representatives of this class are Random Optimization, Simulated Annealing, Tabu Search, and Hill Climbing.
Evolutionary algorithms  Subclass of the Evolutionary Computation, which is a subclass of the Monte Carlo Algorithms uses a process of natural selection and (biology) evolution to find the best solutions over multiple generations. In each evolution, there are several steps to alter and generate new solutions from old ones. We can find strategies like Genetic Algorithms, Evolutionary Programming, Genetic Programming, and Evolution Strategy.

Swarm Intelligence Also a subclass of the Evolutionary Computation. This class of strategies is the collective behavior of decentralized, and self-organized systems (agents), often inspired by natural systems (ant colonies, fireflies, flocks of birds, etc.). Each individual follows simple rules, maintaining their behavior in local interactions, the group as a whole then exhibits complex behavior. Best known algorithms of this class are Ant Colony Optimization, Particle Swarm Optimization, and Artificial Bee Colony Optimization.

2.5.2 Genetic Algorithm

Genetic Algorithm Optimization (GAO) is a robust and stochastic metaheuristic inspired by concepts from natural selection, genetics, and evolution. It efficiently optimizes problems by following the abstracted processes of natural evolution. [33]

The GAO process begins with the initialization of chromosomes (coded solutions), forming what is known as the population. Each individual (chromosome) in the new population is then evaluated using the fitness function. The best individuals serve as parents to generate new ones through crossover functions, where each function takes parts from each parent to create a new individual. With a certain probability, each new individual may undergo mutation, involving changes to some of its genes (chromosome variables).

At the end of each generation, new individuals are selected, and a certain number of old ones (typically the best ones) are carried over to the next generation. This process repeats in each generation until specific criteria are met.

GAO applies to a variety of optimization problems, thanks to the simple modification of chromosomes. All that is required is to create a fitness function, interpret the problem solution as a chromosome, and define operations such as crossover and mutation.

![Figure 2.3 Parts of the Genetic Algorithm.](image-url)
2.5.3 Tabu Search

Tabu Search (TS) is a metaheuristic developed by Fred Glover in 1986, widely employed in solving a diverse array of combinatorial optimization problems. [34]

To enhance the efficiency of the exploration process, TS not only considers local information but also maintains a memory of previously visited locations. This memory is utilized to determine which steps should be avoided to avoid short cycles, hence the name ‘Tabu’.

When TS encounters certain local optima, it cannot linger there due to constraints imposed by the Tabu set of moves. This motivates the strategy to actively seek the next interesting regions within the solution space.

![Figure 2.4 Tabu Search in the maze (after finding the optimum 0, it continues with the search).](image-url)
2.5.4 Simulated Annealing

Simulated Annealing Optimization (SAO) is a probabilistic metaheuristic introduced in 1983 by Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. It draws inspiration from the emulation of a physical process, where a highly heated solid matter is gradually cooled down. The fundamental concept is that the repeated heating and cooling of the material improves its quality.

The process is initiated by setting the temperature variable to a high value (the performance of the strategy is highly dependent on this parameter). Optimization then proceeds by progressively lowering the temperature, thereby reducing the exploration range (acceptance of worsening solutions) and intensifying the exploration of discovered (promising) regions. [35]

![Figure 2.5 Convergence of the Simulated Annealing](image)
2.5.5 **Ant Colony**

Ant Colony Optimization (ACO) metaheuristic is the extension of the Ant System introduced by Marco Dorigo in the early nineties as a new novel heuristic inspired by the behavior of real ant colonies.

ACO can be applied to combinatorial optimization problems with a *finite set* of possible solutions. The ACO starts with a set number of *ants* (agents) finding their path through a graph representation of the optimization problem. Each ant’s path (each node) contains a certain intensity of *pheromone*. Newly placed pheromone is added to the already present one which strengthens it. After every run, every node’s pheromone is slightly reduced (evaporated) to mimic the real-world scenario. Not frequently used paths will contain less pheromone and more frequently used paths will contain more. This information is used by each ant, to decide which next node (path) to take. In the end, the strongest trait in the graph is considered as the shortest path, hence the best solution.

The *evaporation* of pheromone tries to solve the problem with premature convergence in a sub-optimal solution. Changing the amount of evaporating and placing pheromones highly affects the performance of this metaheuristic. The idea is quite simple, if there is an obstacle on the way, the path that takes more ants likely is the shortest as shown in the following picture (more ants; stronger pheromones).[37]

![Figure 2.6 Ant Colony Optimization process.](image-url)
2.6 Hyper-heuristics

The prefix ‘hyper’ denotes a level above or beyond. In the realm of heuristics, we refer to the hyper-heuristic as a higher-level approach that works with a set of lower-level heuristics rather than directly addressing the specific problem and can change its hyper-parameters to achieve better results rather than being hard-wired.

This approach automates the selection, combination, generation, and adaptation of low-level heuristics. The concept is straightforward; every optimization approach has its strengths and weaknesses, and determining the most effective way to utilize them can lead to a highly-performing optimization tool.[9]

In further chapters, I will also introduce the hyper-heuristics used in this work (there are more available in this project, all working and ready to be used), created for the CHeSC2011 competition. These hyper-heuristics can be run, and their results can be evaluated (using the Unit metric (Chapter 3) and F1 metric (Section 4.1) on problem instances provided in the HyFlex framework.

2.6.1 Hyper-heuristics Classification

![Figure 2.7 A classification of hyper-heuristic approaches according to two dimensions (feedback, nature of heuristic search space).[9](image)

<table>
<thead>
<tr>
<th>Feedback</th>
<th>Nature of heuristic search space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online learning</td>
<td>Construction heuristics</td>
</tr>
<tr>
<td>Offline learning</td>
<td>Perturbation heuristics</td>
</tr>
<tr>
<td>No learning</td>
<td></td>
</tr>
</tbody>
</table>

Nature of the heuristics search space  According to the nature of the search space there are

- **Heuristic selection** Choosing already existing heuristics.
- **Heuristic generation** Generating new heuristics from the components of existing ones.
- **Construction heuristics** These methods consider partial candidate solutions, where one or more solution components are missing, and during each iteration, they try to extend them.
- **Perturbation heuristics** These methods evaluate entire candidate solutions, and during each iteration, they try to modify and alter existing solution components.

Sources of feedback information  Using some feedback from the search process.

- **Online learning** The learning occurs as the algorithm solves a problem instance
Offline learning Learning is derived from the accumulated knowledge of a set of training instances, to generalize to solve unseen instances.

No learning No learning takes place.

2.7 Optimization Frameworks

An optimization framework can be interpreted as a systematic approach or an environment implementing a set of software tools for developing, implementing, and solving various optimization tasks. This can involve tools for heuristics, metaheuristics, and even hyper-heuristics.

Typically, there are clearly defined objective functions and constraints for various problem instances across many problem domains. These frameworks can evaluate obtained results using metrics, which are used to compare performances among implemented algorithms or heuristics (including metaheuristics, hyper-heuristics, etc.).

2.8 Metric

Metrics are used to formally define the distance among elements of a certain set $X$. Metric space $M$ is defined as a vector $(M, \psi)$, where $X$ is any non-empty set of elements, and $\psi$ is a metric mapping $\psi : X \times X \rightarrow \mathbb{R}$ defined (behavior and properties) by the following axioms $(\forall x, y, z \in M)$ [38]

Non-negativity $\psi(x, y) \geq 0$

Symmetry $\psi(x, y) = \psi(y, x)$

Triangle Inequality $\psi(x, z) \geq \psi(x, y) + \psi(y, z)$

Identity $\psi(x, y) = 0 \iff x = y$
Chapter 3

Unit Metric

Before introducing frameworks and hyper-heuristics, I will describe a metric and visualizations I’ve implemented to enhance the understanding of results used for comparing algorithms and heuristics (both metaheuristics and hyper-heuristics) in this work.

3.1 Definition

This metric is to provide a unit way that offers easily understandable information about how close or far the result is from the optimal solution.

The mapping of results is done on the interval between the optimal and easily obtained results (using the Greedy algorithm). The mapped value indicates the quality for three aspects: experiment instance, problem domain, and experiment. The Unit metric requires metadata, where information about each problem domain instance is stored. The metadata includes the optimal result value, greedy result value, random result value, and additional details like the size of the problem instance.

3.2 Quality of the Results

The mapping of the results is done on the \((0, 1)\) interval between the easily obtained results (using the Greedy algorithm) and optimal results. This provides a means to assess the quality of the algorithm (heuristic, metaheuristic, or hyper-heuristic).

In the Unit metric, an optimal solution is represented by the number 1, and an easily accessible one by the number 0. All values between them indicate a proximity to either side. The further it is from the number 1, the farther it is from the optimal solution. Conversely, the closer it is to the number 0, the farther the resulting solution is from the optimal solution.

Additionally, comparing the quality of different algorithms on the same instance of the problem domain becomes possible. These numbers reside on the same interval, allowing us to estimate their proximity to each other and determine which one is closer to the optimal solution. Achieving an optimal solution is the shared objective of all algorithms.

3.3 Metadata

The Unit metric utilizes extra information related to problem instances, including name, instance size, optimal solution objective value, and objective values for both greedy and random solutions.
The optimal solution objective values are in this case known, while the medians of 1000 runs were employed for the objective values of the greedy and random searches. The metadata generation was done by the SEAGE framework [Chapter 5] leveraging its preexisting heuristics and data retrieval capabilities.

The generated metadata is also shared with the HyFlex framework [Chapter 4], enabling the utilization of the Unit metric.

**Code listing 3.1** Metadata of the TSP problem domain instance

```xml
<Instance greedy="4" id="rm4" optimum="4" random="6" size="4"/>
```

**Code listing 3.2** Metadata of the SAT problem domain instance

```xml
<Instance greedy="13" id="uf100-01" optimum="0" random="55" size="100"/>
```

**Code listing 3.3** Metadata of the FSP problem domain instance

```xml
<Instance greedy="8" id="rma03_03_01" optimum="8" random="9" size="3"/>
```

**Code listing 3.4** Metadata of the JSP problem domain instance

```xml
<Instance greedy="84" id="ft06" optimum="55" random="86" size="36"/>
```

### 3.4 Instance Equation

The equation for the Unit metric is quite simple; the more challenging part lies in obtaining metadata. It involves knowing the optimal solution (objective value) for each instance of the problem domain and an easily accessible solution (greedy solution). Once we have all these components, the equation is as follows, where $P_i$ represents problem instances, and $P_{ij}$ is a specific problem instance.

$$U_{P_{ij}} = 1 - \frac{f_{P_i}(solution_{P_{ij}}) - optimum_{P_{ij}}}{greedy_{P_{ij}} - optimal_{P_{ij}}} \in (0, 1) \quad \text{where } f_{P_i} \text{ is obj. function of } P_i$$  \hspace{1cm} (3.1)

![Figure 3.1](image)

**Figure 3.1** The Unit metric (score) of an imaginary algorithm’s solution for a certain problem instance.

When dealing with an optimization problem, there is typically an initial solution with a certain quality. It would be beneficial to have an easy-to-understand method to see how much the algorithm/heuristic improved this solution. That’s where $B_{P_{ij}}$ comes in.

$$B_{P_{ij}}(finalSolution_{P_{ij}}) = U_{P_{ij}}(initSolution_{P_{ij}}) - U_{P_{ij}}(finalSolution_{P_{ij}}) \in (0, 1)$$  \hspace{1cm} (3.2)
3.5 Problem Equation

The previous equation evaluates the solution for a single instance. To obtain the metric for the entire domain, the Unit metric needs to be separately calculated as $U_{P,j}$ for each problem instance. The weighted mean is employed because instances vary in size, and the $U_{P,j}$ of larger ones should carry more weight than those of the smaller ones.

$$U_P = \frac{\sum_{i=1}^{[P]} w_{P,j} U_{P,j}}{\sum_{j=1}^{[P]} w_{P,j}} \in [0, 1) \quad \text{where } P_i \text{ are instances of } P, w_{P,j} \text{ is size of } j\text{th instance} \quad (3.3)$$

3.6 Experiment Equation

The final step of the Unit metric involves evaluating the entire experiment, considering the results from each problem domain. The evaluator calculates the mean of all domains, denoted as $U_P$, used in the experiment.

$$U = \frac{1}{|P|} \sum_{i=1}^{[P]} U_{P,i} \in [0, 1) \quad \text{where } P \in \{ \text{SAT, TSP, JSP, FSP} \} \quad (3.4)$$

3.7 Scorecard

All the evaluated results from the algorithm run are organized within a structure referred to as a "scorecard". This term draws inspiration from the CHeSC2011 competition (Chapter 4), which employed a similar structure to track hyper-heuristics results for each instance. In this project, however, the card doesn’t contain the results themselves; rather, it gathers the evaluated Unit scores. Over the years, the format has changed, but for this work, I’ve re-implemented it into a JSON format for easier storage in the database. The initial section of this scorecard includes sets of instance-evaluated results based on the Unit metric, with a distinct set for each problem domain. Subsequently, there is a set containing evaluated results across the problem domains, followed by the evaluated results for the entire experiment. This format is straightforward and is used in both frameworks, each employing a distinct approach for storing these data, as discussed in the following chapters.

Code listing 3.5 Preview of algorithm’s scorecard

```json
{
    "algorithmName": "algorithm1",
    "scorePerInstance": {
        "problemDomain1": {
            "instance1": 0.1,
            "instance2": 0.6
        },
        "problemDomain2": {
            "instance1": 0.1,
            "instance2": 0.0
        }
    },
    "scorePerProblem": {
        "problemDomain1": 0.35,
        "problemDomain2": 0.05
    },
    "totalScore": 0.2
}
```
### 3.8 Heatmap Table

The heatmap table is a visual representation of experiment results where the performance is color-coded. A spectrum ranging from green (indicating better performance) to red (indicating poorer performance) is employed, with yellow representing mid-range. Four thresholds (0.5, 0.75, 0.98, and 1.0) define the color transitions.

![Figure 3.2 Heatmap color range.](image)

For values between 0.0 and 0.5, the intensity of red varies. A value of 0.0 is represented as fully red $RGB(140, 0, 0)$, transitioning to $RGB(250, 0, 0)$ at 0.5. The 0.75 threshold is visualized as $RGB(255, 250, 0)$, while the 0.98 threshold is light green $RGB(1, 150, 32)$, and 1.0 is slightly darker green $RGB(1, 120, 32)$. The choice of colors within these ranges is determined using a sequential color map. This color-coded representation offers a quick and intuitive assessment of performance variations across the experimental results.

There are two types of heatmap tables: the first one displays information about the problem instance results on each problem domain, and the second one shows problem scores across all problem domains.

![Table 3.1 Problem instances heatmap table preview.](image)

![Table 3.2 Problem domains heatmap table preview.](image)
Chapter 4

HyFlex

HyFlex is a framework developed for the CHeSC2011 competition [11], designed to assist (and assist with) the implementation, testing, and comparison of iterative hyper-heuristics. Its concept divides heuristic search into two components. The first one is specialized for algorithms (hyper-heuristics), while the second one is problem-oriented. This can be viewed as a domain barrier between problem-oriented heuristics and hyper-heuristics.

The framework is written in the Java programming language. From a more abstract perspective, just one class needs to be developed by the author of the new hyper-heuristic. This class is named 'HyperHeuristic', which then uses the 'ProblemDomain' class. However, the author does not have any information about the domain itself, only about lower-level algorithms. The HyFlex provides random initialization, four groups of low-level heuristics (Crossover, Mutation, Ruin-recreate, and Local search), two parameters for the intensity of mutation and depth of local search, and an easily accessible population of solutions. [12]

4.1 F1 Metric

The Formula 1 points system was used during the international challenge CHeSC2011 to evaluate the success of individual participants. The idea behind it is as follows: the top eight racers (participants) receive 10, 8, 6, 5, 4, 3, 2, and 1 points, respectively, based on their finishing positions in each race lap, while the rest receive nothing. A racer’s final score is the sum of the points obtained in each round. If there are ten laps, for example, the maximum possible score is 100 points.

In the case of a tie, the points for a particular position are evenly distributed among all tied racers, thus maintaining the total point count for each round. If a tie persists in the final score, the overall number of first-place finishes is considered. If there is still a tie, the number of second-place finishes is taken into account, and so forth. [11]

4.1.1 Metric’s Drawbacks

The issue is not its implementation, but its use for evaluating the quality of solutions for individual heuristics, metaheuristics, and hyper-heuristics. This is caused by its nature. It cannot evaluate results independently. Therefore, it requires multiple participants, but at the same time, the overall evaluation of individuals varies based on their numbers.
4.2 Contribution

In my previous work, I utilized and extended this framework for a significant reason: the reproducibility of results, source codes, and papers describing all the hyper-heuristics (HHs). Multiple hyper-heuristics with stable backgrounds (participating in a global competition) and papers describing the ideas behind them were already implemented.

For anyone seeking to evaluate the performance of a newly developed algorithm, all that needs to be done is to easily compare it with others. This comparison has been implemented in HyFlex (using the $F_1$ metric). However, using this metric, the received evaluation can only be used to rank the algorithms based on their results in one run (competition).

This limitation was overcome with the introduction of the Unit Metric and is now further addressed through a new method of visually representing the results of each algorithm, thanks to the heatmap on both problem domains and their instances. This significantly improves the understanding of the quality of results for each hyper-heuristic.

4.3 Evaluator

Thanks to the previous work on this framework, I was able to expand the old ($F_1$) metric with new Unit metric by implementing a new evaluator. This new evaluator follows the same rules and approaches, ensuring that the reconstruction is consistent with the CHeSC2011 competition.

The results are stored in the form of scorecards, which are then evaluated and compared with the rest of the hyper-heuristics. The only difference in the competition lies in the evaluator used for this final step. To enhance clarity and help with understanding the results, I have implemented a heatmap that employs colors to distinguish the quality of the outcomes. In the following picture, you can see the diagram of this evaluator in the HyFlex.

![Figure 4.1 Diagram of HyFlex with implemented evaluator.](image)

**Problem Domain Instances Selector** In this part, the user modifies the run of the hyper-heuristic—specifying the duration, the number of repetitions, the problem domain, and the directory in which to store the results via the tag parameter.

**Hyper-heuristic Runner** In this part, the user’s selected hyper-heuristic is run on a given set of problem domains.
Result Writer The previous section presents the results of hyper-heuristics on a given set of problem domain instances, and the Result Writer structures them into a strict scorecard format. These scorecards are then stored inside a specified directory for evaluation.

Directory The user-defined directory is where all the results of the experiment are stored.

Evaluator The evaluator evaluates the results to compute the specified metric. In HyFlex, there are two metrics: the F1 metric, which was used in the CHeSC2011 competition (which requires results from other hyper-heuristics), and the Unit metric introduced in my previous work.

Leader-board The result itself (heatmap).

4.3.1 Project Run

A more detailed guide on building and executing the project is available in the Git repository \[39\], particularly in the README.md file. The following code is used to run an experiment with a given hyper-heuristics, timespan, and number of repeats, with an experiment tag on a set of the problem domains (CHeSC2011 instances).

Code listing 4.1 Example command of HyFlex experiment run

```
#!/bin/bash
./scripts/run.sh competition -h ISEA -h LeanGIHH -t 30 -p SAT -p TSP -n 1 --id isea_gihh_exp
```

The previous command executes the HyFlex experiment with specified hyper-heuristics, and the resulting data is saved to a file, as depicted in Figure 4.1. To obtain the leaderboard, which includes the evaluated results using the unit metric and generates the heatmap, use the following command.

Code listing 4.2 Example command of HyFlex evaluation

```
#!/bin/bash
./scripts/run.sh evaluation --id isea_gihh_exp -H -I
```

This command assesses the experiment results and generates a unit-metric-scores.json file. It includes a scorecard for each used hyper-heuristic, a heatmap SVG image for all problem domains (-H parameter), and a heatmap SVG image for instances of each problem domain (-I parameter).

4.3.2 Hyper-heuristics Leader-board

With an understanding of how the Unit metric operates and how the evaluator is utilized in the HyFlex framework, I tested all the existing hyper-heuristics on CHeSC2011 instances \[7.1\]. Thanks to the newly developed heatmap, I can now present the calculated scores for each hyper-heuristic using our Unit metric.
4.4 Selection of HyFlex Hyper-heuristics

The decision of which hyper-heuristic to use in this work has been primarily based on two criteria: effectiveness and ease of re-implementation in the SEAGE framework (PHUNTER’s 2k lines of code proved unbearable). These criteria are met by LeanGIHH, VNS-TW, and ISEA (more about them in the sixth chapter).
SEAGE is an optimization framework introduced as a dissertation (2011) project by Ing. Richard Málek at FEL CTU in Prague.

This framework provides a library of optimization problems (both discrete and continuous) and implementations of heuristics, metaheuristics, and now hyper-heuristics, forming the core of the SEAGE project. Any implemented algorithm or heuristic (whether meta or hyper) can be used as-is, without the need to utilize any other framework features. SEAGE also offers a database for keeping good solutions to start the exploration with and keeping good algorithm parameters from all the previous experiments, making it possible to better understand how the algorithms run, which parameters seem to be the best, and more.

SEAGE’s architecture can be interpreted as a set of layers, each with a specific main task. The upper layers directly utilize the layers below them. For instance, the Experiment layer implements a way to run algorithms/heuristics on a problem, set parameters, work with results, etc. Defining and evaluating experiments is the primary focus of this project and leads to hyper-heuristics.\[40\]
5.1 Contribution

In my previous work, [8], I implemented a new evaluator and metric, thanks to which I illustrated that even metaheuristics, though less sophisticated than hyper-heuristics, achieved comparable results. After implementing a relatively straightforward hyper-heuristic (named basic-hh) in the SEAGE framework, its performance was found to be quite close to that implemented in HyFlex, several times (still lacking behind the best metaheuristic).

Since my bachelor’s thesis, I’ve been actively working on SEAGE, dedicating efforts to further develop, explore, and expand the possibilities of SEAGE. This effort contributed to a more insightful visual representation of each experiment’s results through a heatmap—a table featuring color-coded indications of result quality. Additionally, we have moved from the XML data format to JSON, successfully addressed and expanded the number of problem domains with FSP and JSP, and enhanced the usability of metaheuristics in these domains.

5.2 Coverage of Optimization Problems

The current coverage of optimization problems for each algorithm/metaheuristic is as follows (— means not implemented, while anything else indicates partial or full implementation).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Traveling Salesman</th>
<th>Satisfiability</th>
<th>Quadratic Assignment</th>
<th>Jobshop Scheduling</th>
<th>Flowshop Scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant Colony</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Firefly Algorithm</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Genetic Algorithm</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Particle Swarm</td>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Simulated Annealing</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Tabu Search</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 5.1 SEAGE’s implementation matrix. [40]

5.2.1 Evaluator

Before my work, the solution quality achieved by metaheuristics was represented as an objective value, specific to each problem domain. These results were not easy to understand and needed a better metric, hence the Unit metric presented and implemented in that thesis.

In the image of the evaluator diagram provided below you can see the most important part of the SEAGE evaluator for this work, which takes care of setting, running, and evaluating the metaheuristics and hyper-heuristics results. Instead of complex and not-so-intuitive handling with data as in HyFlex, this implementation uses the full potential of the SEAGE framework, having the data stored in a database, from which it’s easily accessible and handled.
Problem Domain Instances Selector  This part is similar to that in HyFlex; the user modifies the run of the metaheuristic or hyper-heuristic, specifying the duration, the number of repetitions, the problem domain, and the tag to indicate the experiment.

Algorithm Runner  Instead of solely having hyper-heuristics to run, SEAGE also provides the option to use metaheuristics themselves in experiments.

Evaluator  The evaluator is utilized during the experiment’s run to evaluate and store data in the database. This data includes information about the solution, a single run of the algorithm, and the entire experiment.

Database  There are three main tables in the database: one for solutions, one for experiment tasks, and one for the experiment itself. An experiment task represents a single run of the algorithm on one instance.

Reporter  When the experiment is completed, the reporter accesses the database and utilizes the experiment results (represented by a tag) to generate a leaderboard in the form of a heatmap SVG image.

Leader-board  The result itself (heatmap).

5.2.2 Project Run

A more detailed guide on building and executing the project is also accessible in its Git repository [1], specifically in the README.md file. The following code is utilized to run an experiment with specified metaheuristics, timespan, number of repeats, an experiment tag, and a set of problem domain instances (which can be defined as a single instance or a set of instances).

The tag represents the experiment results inside the database, making it easier to then obtain and use for either heatmap table generation or just reporting the experiment’s results. The primary purpose of this tag is to aggregate results from multiple algorithm runs inside the heatmap table, enabling a comparison of their performances among each other.

Code listing 5.1  Example command of SEAGE metaheuristics experiment run

```bash
#!/bin/bash
./seage experiment -single -feedback -a TabuSearch -a SimulatedAnnealing -i TSP:hyflex -i FSP:tai100_20_02 -t 30 -n 1 -T exp_name
```
The following code is used to run an experiment with a specified hyper-heuristic, timespan, number of repeats, an experiment tag, and a set of problem domain instances (either a single instance or a set).

Similarly to the previous case, the tag is also used to mark the experiment results inside the database, which are then utilized for heatmap table generation or reporting the experiment’s results. Thanks to the tag, it’s easier to compare multiple algorithms (both metaheuristics and hyper-heuristics) among each other inside the heatmap table.

**Code listing 5.2**  Example command of SEAGE hyper-heuristic experiment run

```
#!/ bin / bash
./ seage experiment - dohh -i TSP : hyflex -i FSP : tai100_20_02 -t 30 -n 1 -T
dohh_exp
```

The previous commands execute the SEAGE experiment with specified algorithms, and the resulting data are stored in the database, as depicted in Figure 5.1. To obtain the leaderboard (heatmap), which includes the evaluated results of all experiments with the same experiment tag and the results evaluated using the unit metric, use the following command.

The following command retrieves all the algorithm’s results from the database with the same tag and generates a heatmap for both problem domains and instances.

**Code listing 5.3**  Example command of SEAGE report experiments

```
#!/ bin / bash
./ seage report -- tag dohh_exp -H -I
```

This command assesses the experiment results, prints the evaluated experiment results, and generates a heatmap encompassing all problem domains (-H parameter) as well as a heatmap for all used instances of each problem domain (-I parameter).

### 5.2.3 SEAGE Metaheuristics Leaderboard

With an understanding of how the Unit metric works, we can assess the performance of each metaheuristic on the CHeSC2011 instances 7.1—determining whether it performs well or poorly. Later in this work, we will explore whether the combined strength is greater, equal, or smaller than that of the individual.

**Table 5.2**  Score heatmap of SEAGE metaheuristics run 120s, 5 repeats.
Before moving on to implementing the best hyper-heuristics from HyFlex, there are several necessities to address first. The HyFlex framework manages memory access for storing solutions, selects and sets up heuristics with parameters provided by hyper-heuristics, and handles the results of heuristics, among other tasks. All of these components were implemented in the work within the SEAGE framework; the metaheuristics operate on a similar basis.

However, what metaheuristics lack is a place for sharing solutions across the runs; there was no array or pool where these algorithms could store, handle, and maintain previous results. The need for storing the parameters of previously run metaheuristics was also addressed through the ‘Feedback Configurator’. In this chapter, I will describe what had to be implemented and then outline the implementation of HyFlex and my hyper-heuristics, DOHH.

6.1 Solution Pool

In the context of hyper-heuristics, a solution pool refers to a collection of potential solutions for a given problem. This pool typically consists of diverse solutions generated by (in the SEAGE random) solution generators, heuristics, metaheuristics, or other algorithms. The hyper-heuristic can then dynamically select, combine, or manipulate solutions to explore different strategies and improve overall performance. The goal is to maintain a variety of solutions to help heuristics, metaheuristics, or other algorithms achieve better performance during the optimization process.

In the HyFlex framework, this structure cannot be directly handled by hyper-heuristics; however, it can still be used to store and retrieve solutions.

In the SEAGE framework, hyper-heuristics directly maintain this structure (SolutionPool class) to hopefully achieve better performance, allowing for more detailed maintenance of solutions within it.

6.2 Feedback Configurator

In the SEAGE framework, there is a configuration module responsible for creating a configuration for a given metaheuristic following a distinct strategy to set their initial parameters. Several strategies exist; for instance, the ‘Default Configurator’ employs configurations that have proven effective through experimentation and utilize a spread parameter to set parameters within a specified range. This strategy is useful for generating well-performing metaheuristic parameters, as observed from previous runs. Another strategy is the ‘Random Configurator’, which sets the parameters randomly and is good for finding new metaheuristic parameters.
However, what sets hyper-heuristics apart and gives them strength is their ability to learn and adapt from previous steps. They achieve this by adapting to the next step, altering hyper-parameters, selecting distinct solutions, and adjusting the parametrization of heuristics/meta-heuristics. For these purposes, I’ve implemented a new ‘Configurator’ called ‘FeedbackConfigurator’.

This module utilizes prior knowledge about the performance of each metaheuristic parametrization, which is stored in a database. Instead of spending time searching for good parameter values, the module aims to make use of those from prior successful runs, thus avoiding unnecessary effort. Similar to the ‘Default Configurator’, it provides the ability to define the spread parameter. When tasked with creating new configurations, it accesses a database, selects a defined number of the best-performing configurations, and, after regenerating each parameter within a specified range, returns them.

6.3 Hyper-heuristics Re-implementation Similarities

In this section, I will outline the similar changes that had to be made in each re-implemented hyper-heuristic. The re-implementation involves taking the strategies (hyper-heuristics) from the HyFlex framework, which utilizes low-level heuristics, and adapting them to employ more complex metaheuristics within the SEAGE framework. More specific details for each one are described later in this chapter.

Thanks to the nature of hyper-heuristics and their implementation in the HyFlex framework, the task of re-implementation in the SEAGE framework proved to be not so difficult. In the HyFlex framework, hyper-heuristics do not know which heuristic they are using, other than their type—whether they are for exploitation (local search) or exploration (crossover or mutation). Similarly, in the SEAGE framework, there are several types of metaheuristics. For instance, Tabu Search can be considered good for exploitation, and others for exploration. With this in mind, I’ve reused the code from the original implementation. The flowchart of this hyper-heuristic does not change; the logic remains mostly the same. Changes are made to allow the use of metaheuristics instead of the originally used heuristics.

**Code** In the SEAGE framework, all hyper-heuristics are located inside the layer called ‘seage-hyper-heuristics’, just like all metaheuristic experimenters.

To run this hyper-heuristic, I had to implement in each hyper-heuristic the Experiment class to pass all the needed parameters to hyper-heuristics, enabling the SEAGE framework to execute it.

The rest of the code is transferred from the HyFlex framework and just following small changes were made to preserve its logic.

**Timespan** In the HyFlex framework, hyper-heuristics are allowed to use two parameters to alter the heuristic: the depth of the search (each heuristic has its purpose, representing timespan) and the mutation parameter (affecting the mutation rate).

Similarly to SEAGE, two parameters can alter the experiment: the timespan for each run and parameter spread (the range for setting metaheuristic parameters). With the right alteration of the previous parameters, we can achieve similar functionality.

One complication arises from the difference between heuristic and metaheuristic. Due to its larger complexity, a metaheuristic needs more time for good performance. In the HyFlex framework, heuristics were given only a fraction of the time for their run (defined by the depth-of-search parameter). To solve this problem, I’ve introduced the algTimeoutMult parameter, which uses the overall timespan and divides it by ten—this is the longest amount of time the metaheuristic can get. The reason for the division is to balance between the smallest possible value for the overall time parameter (which is 1), the possibility of using as much time for
the metaheuristic as possible if it means gaining a better score, and experimental executions that have shown this approach to be better than any other used.

**Configurations** The other mentioned parameter in the HyFlex framework is the *mutation rate*. Similarly, in the SEAGE framework, it can be directly used for similar purposes. Each metaheuristic needs its parameters generated by some *Generator class*, for instance, ‘FeedbackConfigurator’ or ‘DefaultConfigurator’ (the feedback doesn’t have to have the necessary amount of configurations; in that case, the default is used), which utilizes a *spread* parameter. This parameter is used to generate metaheuristic parameters within this range around a certain value given by the ‘Configurator’.

Therefore, I’ve employed this parameter in such a way that it operates similarly and affects the parametrization of metaheuristics through its logic.

The metaheuristics are selected based on their performance, but there is also a need for them to be configured correctly. One of the main focuses of hyper-heuristics is to maintain a set of diverse solutions and an understanding of how to configure the (meta)heuristics themselves. Thanks to the simplicity of the heuristics, there were only two parameters in HyFlex (*depth-of-search* and *mutation rate*), but in the SEAGE, each metaheuristic has different parameters that need attention. The spread itself, as I presented it, significantly modifies how these parameters are generated—not just within a given range but also with specified central values. However, what are these central values?

For the Default Configurator, these are predefined and chosen based on their good performance in past experiments. Yet, hyper-heuristics should utilize configurations they found helpful in their runs, and that’s where the Feedback Configurator comes in handy. After each phase of the experiment, when the set of tasks is completed, the best-performing configuration is stored in the database for the next phases of the hyper-heuristic. Each hyper-heuristic can then employ these previously stored configurations as it sees fit. Re-implemented hyper-heuristics prioritize the best-performing parameters over the default ones, using spread to determine how much they diverge from this value. This process itself can be considered as training on the go (with some hyper-heuristics having a pre-training phase).

**Runs** The final necessary component for a correct run of metaheuristics selected by the hyper-heuristics is tasks. Each time the hyper-heuristic decides to run a specific metaheuristic, a set of tasks is created—the cores in the machine select the number of tasks—and then they are simultaneously executed, each with slightly different parametrization. After these tasks are finished, the hyper-heuristics continues.
6.4 GIHH/LeanGIHH

6.4.1 Motivation

The selection of this hyper-heuristic is based on its performance during the CHeSC2011 competition, where it secured first place. Additionally, its significance in the 2016 case study [13], which resulted in a reduction in complexity, further supports its choice for re-implementation, simplifying the overall process.

6.4.2 Overview

The Adaptive Hyper-heuristic (AdapHH), also known as GIHH, is a hyper-heuristic that won the CHeSC2011 competition. The authors, Mustafa Misir, P. De Causmaecker, G. Vanden Berghe, and K. Verbeeck, introduced a novel selection hyper-heuristic with several adaptive features to address the management of different sets of heuristics.

The proposed approach intelligently selects heuristics, identifies effective heuristic pairs, and dynamically adjusts the parameters of specific heuristics online. Additionally, the authors developed an adaptive list-based threshold-accepting mechanism, enabling the system to decide whether to accept the solutions generated by the selected heuristics or not.

Among the key features is the Adaptive Dynamic Heuristic Set (ADHS) which assesses the performance of each heuristic over several iterations, and maintains only the best-performing heuristic.

Relay hybridization implements a simple relay hybridization approach to determine effective pairs of heuristics, and maintains a list for each heuristic.

Heuristic parameter adaptation dynamically adapts parameters of heuristics (intensity of mutation and depth of search) using a reward-penalty strategy.

Adaptive Iteration Limited List-based Threshold Accepting (AILLA) represents a move acceptance mechanism that decides whether to accept solutions generated by heuristic and dynamically determines the threshold level using fitness values of the previously found best solution.[42]

The LeanGIHH hyper-heuristic is a simplified variant of GIHH (Adap-HH), which emerged as the winner of the CHeSC2011 competition. It was derived from the original through Accidental Complexity Analysis (ACA), a technique aimed at reducing algorithmic complexity without compromising performance. The implementation of LeanGIHH successfully reduced the original 2324 lines of code to a mere 288 lines.[13]
6.4.3 Flowchart

Figure 6.1 LeanGIHH flowchart.
6.4.4 Re-implementation

**Code** The whole code for the LeanGIHH hyper-heuristic is located in two files, where the first one contains the whole logic, and the other one contains the implementation of a circular buffer.

The circular buffer is a generic limited capacity FIFO queue. Elements are pushed at the front, and when the buffer exceeds its capacity, the last element is dropped. This buffer is used for the selection of (meta)heuristic logic.

**Timespan** The maximal depth-of-search is initialized inside the constructor and then utilized throughout the hyper-heuristic.

The timespan is calculated from the 'val' variable, which holds the same value for both the depth-of-search and intensity of the mutation parameter for each metaheuristic.

**Configurations** The intensity of mutation that defines the spread for each metaheuristic parameter is stored inside the 'val' array.

The configurations for each task are generated by FeedbackConfigurator. If there are not enough configurations in the database, it is then filled with configurations from DefaultConfigurator.

The training for obtaining the best metaheuristics configuration is on the go, during each phase of the hyper-heuristic.

**Runs** The tasks are associated with solutions that are required by the metaheuristic according to their nature. For instance, Tabu Search and Simulated Annealing require one solution, while Ant Colony or Genetic Algorithm require more solutions.

The LeanGIHH algorithm provides one or two solutions, depending on the nature of the selected heuristic. What differs in SEAGE from the HyFlex framework is that the exploration-type metaheuristic requires more solutions. This has been addressed by using the provided solutions and filling the remainder with random solutions using the IProblemProvider class.

The number of configurations, hence tasks, is defined by the capability of the system, with the default value set to 6.
6.5  VNS-TW

6.5.1  Motivation

The choice for this hyper-heuristic is because of its performance shown in Table 6.1. Following the application of a newly introduced Unit metric, it achieved the fourth position, behind the GIHH/LeanGIHH and NAHH hyper-heuristic. Furthermore, the decision to select this hyper-heuristic also considered its simpler implementation, especially when compared to other HyFlex hyper-heuristics that achieved similar results (PHunter, for instance, has an unbearably around 2k lines).

6.5.2  Overview

The Variable Neighborhood Search (VNS) Hyper-heuristic based on dynamical local search strength adjustment and the use of a population archive secured the 4th place in the CheSC2011 challenge. The authors, Ping-Che Hsiao, Tsung-Che Chiang, and Li-Chen Fu are members of the Department of Computer Science and Information Engineering at National Taiwan University.

VNS-TW proposes an approach comprising four key steps: shaking, local search, environmental selection, and periodical adjustment. It starts with a random initial solution and involves selecting low-level heuristics (LLHs) for shaking and local search phases, followed by environmental selection to determine which solution in the population to replace.

Shaking uses mutation and ruin-recreate heuristics on the base solution to generate a new solution, with a tabu mechanism to avoid frequently applying poor heuristics.

Local search heuristics are applied iteratively. Heuristics are ranked by their performance, and the local search continues until all heuristics reach the lowest rank or a set number of non-improving iterations are reached.

Environmental selection decides whether to replace the current solution with the new solution based on their fitness values. It also includes a tournament selection process to maintain a diverse population.

Periodical adjustment adjusts parameters like the number of non-improving iterations allowed in the local search step, based on the search progress.

The pros of this hyper-heuristic lie in its flexible adaptation to various problem domains, Tabu mechanism, and heuristic ranking, contributing to effective search. Additionally, it features a population archive that maintains diversity in solutions. However, there are a few cons, including the complexity of managing multiple phases and dynamically adjusting parameters.
6.5.3 Flowchart

![Flowchart diagram]

**Figure 6.2** VNS-TW flowchart.
6.5.4 Re-implementation

**Code** Similarly to the previous hyper-heuristic, this one is composed of two files, where one file contains the entire logic and the other, 'HeuristicCluster'.

The heuristic cluster is a structure that helps with the selection of (meta)heuristics, keeping track of time, values, and their type, and also implements the logic for (meta)heuristic selection (wheel selection). The VNS-TW has two heuristic clusters, one for the exploitation (meta)heuristic and the other for exploration (meta)heuristics.

**Timespan** The code itself feels straightforward with its logic, but some areas appear to be unfinished. For instance, the depth-of-search is initialized with a value of 0.2, and the only changes (increasing by 0.2) to this parameter are made when the current solution fitness is the same as the previous best. There is no decrease in this value, so after a certain time, it reaches the maximal allowed depth-of-search, which is 0.8, and then remains at this value.

Similarly, with the LeanGIHH hyper-heuristic, the depth of search defines what fraction of the total allowed time is used in its tasks.

**Configurations** The unfinished nature of this hyper-heuristic is evident even here, where the mutation rate, used to define the spread for both 'FeedbackConfigurer' and 'DefaultConfigurer', is set once and never changed. I've set this value based on the performance in a few experiments, although the difference wasn’t that significant; the value is 0.5.

The pre-training of metaheuristic parameters is conducted as the initial step following the initialization of VNS-TW hyper-parameters. During this phase, the hyper-heuristic performs several tests to gain a better understanding of the building blocks it needs to utilize. As a byproduct of these tests, it also learns and stores the best-performing metaheuristic parameters for the following main search phases.

**Runs** The hyper-heuristic provides only one solution for both exploration and exploitation (meta)heuristics. Consequently, the remaining required solutions are filled with random ones, generated by the 'IProblemProvider' class.

The execution of metaheuristics poses a challenge due to the possible overwhelming time complexity. In various sections of the hyper-heuristics, there is no check to verify if there is still time remaining. Consequently, the loop, expecting fitness improvement, may end up in an endless loop. I addressed these issues by introducing a method called 'isStillTime', which returns true if there is time remaining and breaks the loops otherwise.
6.6  ISEA

6.6.1  Motivation

Last but not least, the motivation for choosing this hyper-heuristic is also based on the explored performance in the 4.2.1 leaderboard, where it secured eighth place. Additionally, this hyper-heuristic holds significance as the only one authored by a member of CTU faculty, specifically the Electrical Engineering. On the other hand, the implementation proved to be quite challenging, as it is located in several files; however, these code segments weren’t overly difficult to re-implement.

6.6.2  Overview

The Iterated Local Search Hyper-heuristic Driven by Evolutionary Algorithm (ISEA) secured 8th place in the CHeSC2011 challenge. The author, Jiří Kubalík, is a member of the Department of Cybernetics at Czech Technical University in Prague.

ISEA is built upon an evolutionary-based iterative local search algorithm called POEMS. POEMS is an optimization algorithm that operates on a single candidate solution, referred to as a prototype (a fixed-length sequence of low-level heuristics (LLHs)), and aims to enhance it through an iterative process. The evolutionary algorithm (EA) is employed in each iteration to explore the most beneficial modifications to the prototype.

The action sequences are used based on their impact on the prototype. There are three types of these actions: variation, mutation, and nop (no operation). The variable length of action sequences allows flexibility. After the EA process, if the modified prototype improves or matches the current one, it becomes the new prototype for the next iteration; otherwise, the prototype remains unchanged. The iterative process finishes after a specified number of iterations.

ISEA maintains a set of three solutions to the given problem: evaluation solution, working solution, and best-so-far solution. The evaluation solution is used for assessing candidate sequences of LLHs. The working solution stores intermediate solutions and the final solution is successfully generated from the initial evaluation solution by applying individual LLHs. The best-so-far solution represents the best solution found so far.

POEMS combines the advantages of both single-state and population-based metaheuristics, exploring a larger neighborhood defined by fixed-length action sequences. This enhances exploration capabilities, making it more effective than standard single-state techniques.

The pros of this hyper-heuristic lie in the effective integration of ILS (Iterated Local Search) and EA for improved solution quality, and the adaptive nature of the algorithm, allowing for dynamic adjustment to different problem instances. On the other hand, there are a few cons in the complexity of understanding and implementing the combination of ILS and EA strategies, and the effectiveness of the re-initialization strategy requires careful tuning. [44]
6.6.3 Flowchart

![Flowchart Image]

Figure 6.3 ISEA flowchart.
6.6.4 Re-implementation

**Code** The ISEA requires more code than the previous two hyper-heuristics, primarily due to increased complexity in logic as it develops its evolutionary algorithm. In total, there are 10 logic files and one for SEAGE purposes—the experimenter. However, thanks to it being spread across several files with a manageable number of code lines, it wasn’t that difficult to re-implement in SEAGE, especially when compared to 'PHUNTER', for instance.

All the files are the same as in the HyFlex implementation with a slight change in the 'EvoCOPSEAGEManager', which was previously the HyFlex manager. However, because it’s no longer in HyFlex, I rewrote the HyFlex-dependent part with the SEAGE ones. This includes the method for running the metaheuristics, generating random numbers, generating initial solutions, comparing solutions, etc.

**Timespan** The timespan is dependent on the 'deptOfSearch' parameter that changes dynamically as the hyper-heuristic progresses. We can observe this change in the 'EvoCOPAction', 'EvoCOPHyperHeuristic' (where I modified the value due to the time complexity of the method for complexity testing), 'EvoCOPLowLeverH', and 'EvoCOPSolution' classes.

**Configurations** The configurations use the 'intensityOfMutation' parameter as a spread parameter. I’ve already explained what this parameter is used for.

Similar to the 'depthOfSearch' parameter, the intensity of mutation also changes dynamically as the hyper-heuristic progresses. This change is implemented in the 'EvoCOPSolution', 'EvoCOPLowLeverH', and 'EvoCOPAction' classes.

Similar to VNS-TW, ISEA also undergoes a pre-training phase (with 'testTimeComplexity' method), during which it attempts to update its knowledge about the problem instance time complexity using each metaheuristic. As a byproduct, a set of best parameters for each metaheuristic is stored in the database for later use.

**Runs** Similar to the previous hyper-heuristic, this one also doesn’t offer more than one solution per run for both exploration and exploitation (meta)heuristics. Based on the required number of solutions, the rest is generated by the 'IProblemManager'.

After this slight change, the process of creating and running tasks remains the same. For each task, the configuration is generated by the 'FeedbackConfigurator' or 'DefaultConfigurator', and then it’s run for a given amount of time. The result of this run is then stored in a defined place in the solution pool.
6.7 SEAGE Hyper-heuristic DOHH

6.7.1 Motivation

Finally, let me present my hyper-heuristic, DOHH (David Omrai Hyper-Heuristic). The motivation behind creating the DOHH was to integrate all the necessary components that had been developed since the creation of the SEAGE framework. Previously, the framework only allowed for the separate use of metaheuristics without a higher-level logic defining their cooperation. In a prior project, I implemented a simple heuristic called 'BasicHH2', which randomly selected the next metaheuristic as the solution. This straightforward hyper-heuristic served as a way to test the functionality of the implemented parts and their potential for metaheuristic cooperation. However, this naive logic didn’t yield impressive scores, at best still quite worse than the best single metaheuristic. This raised the question: How could it be done better? Lacking prior experience, I researched the hyper-heuristics developed in HyFlex. Thanks to the available code and papers describing their logic, I gained insight into how they approach the problem.

Thanks to modern technologies, particularly language models (ChatGPT), I considered using them to gather information from each hyper-heuristic for the best possible approach. However, this task proved too demanding for the models; although they understand how hyper-heuristics operate, it’s only at an abstract level. The generated blueprint for the best possible hyper-heuristic was essentially a blueprint for basic hyper-heuristic logic.

Therefore, I combined the information from HyFlex hyper-heuristics and researched potential reinforcement learning strategies. I aimed to keep this hyper-heuristic as simple as possible, feeling overwhelmed by the complexity of certain HyFlex hyper-heuristics, and concluded with the following solution. I was mostly inspired by the easy flow of the LeanGIHH hyper-heuristic but I used fully my logic.

6.7.2 Overview

The DOHH hyper-heuristic (David Omrai Hyper-Heuristic) draws inspiration from reinforcement learning, specifically from epsilon Thompson Sampling, and also has its implementation of a beta random generator (using the Marsaglia method [45]).

Reinforcement learning (Thompson’s sampling) [46] plays a significant role in deciding which metaheuristic should follow the computation. Another crucial decision is made by closely observing the metaheuristics themselves—considering how well they improve the solution, which metaheuristics perform better after the previous one, etc.

In the following few points, I will introduce the logic behind blocks shown in the flowchart.

Initialization As the first step, DOHH initializes all variables and hyper-parameters, similar to other hyper-heuristics such as solution pool, search depth, and spread.

The size of the solution pool is set to 7, with the first place reserved for the best solution so far, the second for the current best—periodically changing based on the performance of metaheuristics—and the last place reserved for the current task’s solutions. The rest are used by the hyper-heuristic to maintain a diverse population of solutions, which are later employed by the exploration metaheuristics.

The maximal depth of search is deduced from the total timeout, constituting a tenth of this value.

DOHH introduces several new hyper-parameters specific to this hyper-heuristic. These include 'heuristic last improvement', which keeps track of the amount of the last improvement for each metaheuristic; 'tucked turns', which monitors how many turns there were with no
new best solution found; and 'solution reset', which tracks how many runs have passed since the reset of parameters due to the hyper-heuristic being stuck.

Furthermore, there are two matrices, 'next algorithm success' and 'next algorithm failure', which keep track of how well the metaheuristics work together.

Metaheuristic selection The metaheuristic selection is based on several conditions and can be found in the 'selectHeuristic' method. With an 'exploration rate' probability, each step can select just an exploration-type metaheuristic based on the roulette selection. In this selection, the 'selection score' for each metaheuristic is used—better performance in the past results in a higher value. When the exploration rate hasn’t caused the selection of only an exploration-type heuristic, the classic selection based on three parts is used.

The first part is which metaheuristic the epsilon Thompson’s sampling selects. Thompson’s sampling balances exploration and exploitation by estimating the uncertainty of each option and selects actions based on sampling from these (metaheuristics) distributions. The Bayesian approach allows for adaptive decision-making, adjusting choices over time as information about whether the metaheuristic failed (hasn’t produced a better solution) or succeeded arrives. This part contributes 80% of the selection score (value selected through experimental testing).

The second part, with a weight of 5% in the selection score, considers the amount of the last improvement. I aimed to provide a slight advantage to metaheuristics that produced a better fitness update. This value represents the improvement of the current metaheuristic compared to the other metaheuristics.

The third part, with a weight of 15% in the selection score, takes into account the previous success of the current and previous metaheuristics. This value represents the rate of successful runs compared to all runs.

These three values are then combined into a single value within the range of 0 to 1. This combined value is used in roulette selection to determine the next metaheuristic to be selected.

Metaheuristic configuration The configuration of the selected metaheuristic is quite similar to those used in re-implemented hyper-heuristics, yet it doesn’t blindly select the best configuration from a given range. Instead, the configuration spread dynamically changes to reflect the state of the search, whether it’s stuck, and for how long.

This logic is housed within the 'getHeuristicConfigs' method. The spread for each metaheuristic is located inside the 'heurConfigsSpread' array, which dynamically changes based on the performance of a given metaheuristic. If it performs well, the spread is reduced by 0.5. If the new score remains the same, it’s increased by 0.15, and if the new score is worse, it’s increased by 0.5 (with limits that the spread cannot exceed [0.2, 1.0]).

The best previous parameters are selected if the search is not stuck. Otherwise, with a probability determined by the number of stuck phases, it’s half of the configurations generated by the Default Generator. This approach helps to discover new possible configurations.

The DOHH hyper-heuristic, similar to the LeanGIHH, lacks a pre-training phase and learns on the go.

Performance update The performance update is executed in the 'updatePerformanceMetrics' method, which updates the hyper-parameters of the hyper-heuristic based on its performance. First, it checks whether the hyper-heuristic is stuck more than allowed. Then, based on the improvement or lack thereof, the hyper-parameters of the hyper-heuristic (metaheuristics parameters spread, depth of search, improvement made, etc.) are adjusted. The update is done to try to reduce the time for those metaheuristics that perform poorly and increase the time for those that have potential. The values themselves are selected for a slight update, which may over time yield better performance.
In the end, Thompson’s sampling is updated for the selected metaheuristic, +1 for the success counter if the new fitness is better than the last one, or +1 for the failure counter otherwise.

**Solution diversity** The solution diversity check is done in the 'checkSolutionDiversity' method, and it checks solution pool diversity and decides whether to replace it with new or leave it be. In the case of metaheuristic, its value is stored either way.

The renewal of solutions is triggered when the hyper-heuristic is stuck on maximum runs, and the reset hasn’t been performed a set number of runs.

The exploration metaheuristics store their solutions each time, replacing the worst solution. In the absence of a worse solution, they replace a random solution.

**Accepting new solution** The acceptance of the current solution as the focal point of the next exploration is determined by its fitness value. If the fitness is worse, the decision is based on the worsening factor (within the allowed range) and the necessity for a new solution, particularly in cases where the search is stuck.
6.7.3 Flowchart

![Flowchart Image]

**Figure 6.4** DOHH flowchart.
6.7.4 Experiment Schedule

In the following image, there is a showcase of the DOHH experiment schedule on the hard TSP instance u2152. Each cell contains the name of the used metaheuristic, the achieved score, and the run time.

![Figure 6.5 DOHH experiment metaheuristics schedule on TSP instance u2152 [60 s, 1 repeat].](image)
Chapter 7

Experiments

This chapter serves as a comprehensive showcase of the extensive work on this thesis and examines the experimental evaluation of hyper-heuristics and metaheuristics across various problem domains. The examination focuses on the efficacy of these algorithms in handling challenging instances of several problem domains, also used in the CHeSC2011 competition. The main goal is to determine whether the re-implemented hyper-heuristics within the SEAGE framework, using more complex building blocks (metaheuristics) demonstrate superior performance compared to their implementation in HyFlex where they use simpler building blocks (heuristics).

The experiments are strategically structured into two distinct parts. In the first part, the focus is on gathering material to either accept or reject the hypothesis regarding the performance of hyper-heuristic building blocks.

The second part of the experiments observes whether the implemented hyper-heuristics, within the SEAGE framework effectively use the performance of individual metaheuristics. The main question here is whether these hyper-heuristics can surpass the standalone performance of the used metaheuristics.

The experiments were performed on a SEAGE database, which was initially empty. Initially, the HyFlex hyper-heuristics were executed, followed by the SEAGE metaheuristics, and finally, the SEAGE hyper-heuristics.

7.1 Instances

Thanks to my work on this thesis, SEAGE has expanded its usability from SAT and TSP to two additional problem domains: FSP and JSP. In the first part of the experiments, I will be using three problem domains (SAT, TSP, and FSP) available in both the HyFlex and SEAGE frameworks.

The second part presents all four problem domains (SAT, TSP, FSP, and JSP) for a more comprehensive comparison of hyper-heuristics against metaheuristics.

The problem domains and instances used for this work are as follows.

<table>
<thead>
<tr>
<th></th>
<th>SAT size</th>
<th>TSP size</th>
<th>FSP size</th>
<th>JSP size</th>
</tr>
</thead>
<tbody>
<tr>
<td>instances</td>
<td>pg-525-2276</td>
<td>525</td>
<td>pr299</td>
<td>tai100_20_02</td>
</tr>
<tr>
<td></td>
<td>pg-696-3122</td>
<td>696</td>
<td>usa3509</td>
<td>tai500_20_02</td>
</tr>
<tr>
<td></td>
<td>pg-525-2336</td>
<td>525</td>
<td>rat575</td>
<td>tai100_20_04</td>
</tr>
<tr>
<td></td>
<td>jarv-684-2300</td>
<td>684</td>
<td>u2152</td>
<td>tai200_20_01</td>
</tr>
<tr>
<td></td>
<td>hj4-300-1200</td>
<td>300</td>
<td>d1291</td>
<td>tai500_20_03</td>
</tr>
</tbody>
</table>

*Table 7.1* Instances of each problem domain used in the experiments.
**SAT instances** are represented in conjunctive normal form (CNF), which expresses logical formulas using conjunctions (AND) and disjunctions (OR) in a standard way. This CNF formula is a conjunction of clauses, where each clause is a disjunction of literals, with three literals in each clause in this work. *The size of these instances corresponds to the number of variables.*

**TSP instances** are represented as a complete graph, where cities are nodes, edges represent possible routes between cities, and each edge has an associated cost (distance). These instances are stored as a matrix specifying the distances between pairs of cities, *and the dimension of this matrix corresponds to the size of the instance.*

**JSP instances** are represented by information about the machines, jobs, number, and length of operations. Each job is identified by a unique identifier and priority. Each job contains a set of operations that includes information about their length, number, and associated machine ID. Machines are assigned to each operation and are represented by unique identifiers. Instances in this work are stored as XML files containing all the mentioned information. *The size of these instances corresponds to the number of operations.*

**FSP instances** are simpler variations of JSP instances, and even though they share all the components (machines, jobs, and operations), their representation is much simpler. Each machine has a defined sequence of operations represented by jobs’ unique identifiers. These instances are stored as a matrix containing information about machines and job sequences. *The size of these instances corresponds to the total number of operations on each machine.*

### 7.2 Used Metric

In the evaluation of the (hyper/meta)heuristics performance, three parts of the same Unit metric will be used (more described in [Chapter 3](#)). Firstly, the *instance-specific unit metric* provides insight into how the algorithms deal with individual problem instances.

Secondly, the *problem-specific unit metric* offers a broader perspective on the algorithm’s efficacy across entire problem domains.

Lastly, the *overall unit metric* describes the overall performance across all problem domains, providing a comprehensive measure of the algorithm’s effectiveness.

### 7.3 HyFlex & SEAGE Hyper-heuristics Experiments

The following experiments in this section are designed to compare three *re-implemented* hyper-heuristics in the SEAGE framework with the original version in HyFlex. The goal is to gather information supporting or refuting the hypothesis of whether hyper-heuristics in SEAGE, using metaheuristics, outperform those in HyFlex, which employ simple heuristics.

Experiments vary in duration and repeats of runs on three problem domains (*FSP, SAT, TSP*). Initial experiments focus on overall scores, while the second part examines longer runs with more repeats, *(60s and 120s, 4 and 8 times).* Multiple runs (repeats) are employed to provide algorithms with additional opportunities to discover improved results. Subsequently, after running the algorithms on all problem instances, the results are aggregated using the Unit metric and are later used to generate heatmaps tables, visualizing the overall results. This analysis aims to understand how the hyper-heuristic’s performance is influenced by run duration and repeats, contributing to a comprehensive hypothesis evaluation.
7.3.1 First Experiment [60 seconds, 4 repeats]

In this first experiment, each hyper-heuristic is executed on every available problem domain for 60 seconds, repeated 4 times.

In the FSP and SAT problem domains, the results closely align with the original implementations, although the original still outperforms in all domains, primarily on the TSP.

An anomaly is observed in the SEAGE hyper-heuristics performance on the TSP domain, where it obtains approximately 0.23 in score compared to the 0.8 achieved in the HyFlex framework. This anomaly may be attributed to a potentially sub-optimal implementation of this specific problem domain. Further analysis in the next experiments reveals that this performance is dependent on the quality of results produced by the metaheuristics utilized by the hyper-heuristics in SEAGE.

Despite these variations, my re-implemented hyper-heuristics, with my new DOHH, result similarly in the SEAGE for this specific duration and repeats, and for the VNS-TW hyper-heuristic on the SAT domain, it even outperforms the HyFlex ISEA. It closely follows the performance of hyper-heuristics in HyFlex on the FSP and SAT problem domains.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>FSP</th>
<th>SAT</th>
<th>TSP</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeanGIHH</td>
<td>Steven Adriaensen</td>
<td>0.89643</td>
<td>0.93534</td>
<td>0.83733</td>
<td>0.8897</td>
</tr>
<tr>
<td>VNS-TW</td>
<td>Ping-Che Hsiao</td>
<td>0.88749</td>
<td>0.94141</td>
<td>0.74772</td>
<td>0.85587</td>
</tr>
<tr>
<td>ISEA</td>
<td>Jiri Kubalik</td>
<td>0.88095</td>
<td>0.85982</td>
<td>0.82564</td>
<td>0.85514</td>
</tr>
</tbody>
</table>

Table 7.2 HyFlex hyper-heuristics experiment scores 60 seconds, 4 repeats.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>FSP</th>
<th>SAT</th>
<th>TSP</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>re-VNS-TW</td>
<td>Ping-Che Hsiao / Omrai</td>
<td>0.80993</td>
<td>0.86499</td>
<td>0.24625</td>
<td>0.64039</td>
</tr>
<tr>
<td>DOHH</td>
<td>David Omrai</td>
<td>0.83437</td>
<td>0.81107</td>
<td>0.23673</td>
<td>0.63405</td>
</tr>
<tr>
<td>re-LeanGIHH</td>
<td>Misir/Adriaensen/Omrai</td>
<td>0.80896</td>
<td>0.81862</td>
<td>0.24086</td>
<td>0.62441</td>
</tr>
<tr>
<td>re-ISEA</td>
<td>Jiri Kubalik / Omrai</td>
<td>0.80547</td>
<td>0.78297</td>
<td>0.22167</td>
<td>0.60337</td>
</tr>
</tbody>
</table>

Table 7.3 SEAGE hyper-heuristics experiment scores 60 seconds, 4 repeats.
7.3.2 Second Experiment [60 seconds, 8 repeats]

In the second experiment, run 60 seconds, repeated 8 times, reveals that three SEAGE hyper-heuristics (DOHH, VNS-TW, and LeanGIHH) outperform the ISEA HyFlex hyper-heuristic on the SAT domain. Additionally, on the FSP domain, all SEAGE hyper-heuristics result similarly to the score achieved by the ISEA hyper-heuristic in HyFlex. However, it’s important to note that the TSP domain still exhibits poor performance in the SEAGE framework.

The poor performance on the TSP problem domain persists in all experiments for SEAGE hyper-heuristics. However, the other problem domains show a close resemblance to the original implementation. Thanks to the heatmap table, we can observe similar performance on the SAT problem domain, where both LeanGIHH and VNS-TW outperform the ISEA hyper-heuristic. On the FSP, they yield similar results.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>FSP</th>
<th>SAT</th>
<th>TSP</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LeanGIHH</td>
<td>0.89147</td>
<td>0.93048</td>
<td>0.81059</td>
<td>0.87752</td>
</tr>
<tr>
<td>2</td>
<td>VNS-TW</td>
<td>0.88335</td>
<td>0.94173</td>
<td>0.76902</td>
<td>0.8647</td>
</tr>
<tr>
<td>3</td>
<td>ISEA</td>
<td>0.8818</td>
<td>0.8494</td>
<td>0.79953</td>
<td>0.84358</td>
</tr>
</tbody>
</table>

Table 7.4 HyFlex hyper-heuristics experiment scores 60 seconds, 8 repeats.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>FSP</th>
<th>SAT</th>
<th>TSP</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOHH</td>
<td>0.83296</td>
<td>0.88034</td>
<td>0.75572</td>
<td>0.65534</td>
</tr>
<tr>
<td>2</td>
<td>re-VNSTW</td>
<td>0.81607</td>
<td>0.86987</td>
<td>0.67319</td>
<td>0.64638</td>
</tr>
<tr>
<td>3</td>
<td>re-LeanGIHH</td>
<td>0.81544</td>
<td>0.86101</td>
<td>0.74042</td>
<td>0.63896</td>
</tr>
<tr>
<td>4</td>
<td>re-ISEA</td>
<td>0.81788</td>
<td>0.7861</td>
<td>0.74044</td>
<td>0.61944</td>
</tr>
</tbody>
</table>

Table 7.5 SEAGE hyper-heuristics experiment scores 60 seconds, 8 repeats.
7.3.3 Third Experiment [120 seconds, 4 repeats]

In the third experiment, with doubled duration (120 seconds) and 4 repeats once again, the poor performance of SEAGE hyper-heuristics on the TSP domain persists. However, on the SAT domain, we observe them outperforming the ISEA hyper-heuristic implemented in HyFlex. On the FSP problem domain, all SEAGE hyper-heuristics are very close to the performance of the worst HyFlex hyper-heuristic, ISEA.

Thanks to the heatmap table, it’s easy to see that the performance resembles the original implementation. In the SAT problem domain, both VNS-TW and LeanGIHH outperform the ISEA hyper-heuristics with a similar ratio. The results in the FSP problem domain are also closer to the original implementation. Therefore, we can conclude that hyper-heuristics utilizing metaheuristics do not significantly yield better or worse problem solutions. Perhaps their complexity contributes to a slightly poorer performance, but this is also dependent on how well the re-implementation is implemented.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>FSP</th>
<th>SAT</th>
<th>TSP</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LeanGIHH</td>
<td>0.89361</td>
<td>0.94168</td>
<td>0.84317</td>
<td>0.89282</td>
</tr>
<tr>
<td>2</td>
<td>VNS-TW</td>
<td>0.89386</td>
<td>0.94881</td>
<td>0.7789</td>
<td>0.87386</td>
</tr>
<tr>
<td>3</td>
<td>ISEA</td>
<td>0.8861</td>
<td>0.87204</td>
<td>0.83731</td>
<td>0.86515</td>
</tr>
</tbody>
</table>

Table 7.6 HyFlex hyper-heuristics experiment scores 120 seconds, 4 repeats.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>FSP</th>
<th>SAT</th>
<th>TSP</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOHH</td>
<td>0.85554</td>
<td>0.88727</td>
<td>0.29641</td>
<td>0.66641</td>
</tr>
<tr>
<td>2</td>
<td>re-VNSTW</td>
<td>0.83824</td>
<td>0.86252</td>
<td>0.27963</td>
<td>0.65886</td>
</tr>
<tr>
<td>3</td>
<td>re-LeanGIHH</td>
<td>0.84296</td>
<td>0.87496</td>
<td>0.29802</td>
<td>0.65865</td>
</tr>
<tr>
<td>4</td>
<td>re-ISEA</td>
<td>0.85293</td>
<td>0.80964</td>
<td>0.2988</td>
<td>0.64045</td>
</tr>
</tbody>
</table>

Table 7.7 SEAGE hyper-heuristics experiment scores 120 seconds, 4 repeats.
7.3.4 Fourth Experiment [120 seconds, 8 repeats]

In the final fourth experiment, which centers on comparing HyFlex and SEAGE hyper-heuristics with a duration of 120 seconds and 8 repeats, a consistent pattern of poor performance is observed for SEAGE hyper-heuristics on the TSP problem domain, despite a slightly improved score. Once more, there is a notable out-performance of SEAGE hyper-heuristics over the ISEA hyper-heuristic from the HyFlex framework on the SAT problem domain. The scores of all SEAGE hyper-heuristics on the FSP problem domain are close enough to each other and comparable to the worst-performing HyFlex hyper-heuristics, ISEA.

In the heatmap, we observe similar performance compared to the previous experiment with 4 repeats. The results of the re-implemented hyper-heuristics are once again poor, yet the results for the rest of the problem domains show comparable quality. Almost all hyper-heuristics perform very closely to the results of the original implementation in both the FSP and SAT problem domains, with a few instances of better results in the SAT problem domain, where all re-implemented hyper-heuristics outperform the original ISEA hyper-heuristic. Therefore, we can also conclude from this experiment that hyper-heuristics utilizing metaheuristics instead of heuristics do not significantly yield better or worse problem solutions, even though using heuristics seems to be a better approach in this case.

Table 7.8 HyFlex hyper-heuristics experiment scores 120 seconds, 8 repeats.

Table 7.9 SEAGE hyper-heuristics experiment scores 120 seconds, 8 repeats.

For a more detailed understanding of each hyper-heuristic performance on each instance within the used problem domains, I conduct a thorough examination of the previous results.
7.3.5 SAT Problem Domain [120 seconds, 8 repeats]

Analyzing the results from the hyper-heuristics re-implemented in the SEAGE framework, it is evident that they yield comparable scores compared to their original counterparts in the HyFlex framework. Particularly, the performance of the re-implemented ISEA hyper-heuristic outperforms or achieves comparable results in five out of five instances. This suggests that implementation of the SAT problem domain and associated metaheuristics in SEAGE can outperform the heuristics in HyFlex, although not to a significant, yet still close extent. Additionally, my new hyper-heuristic outperforms the ISEA hyper-heuristic in two out of the five instances, showcasing its effectiveness in certain problem domains.

Thanks to the heatmap table, we can observe that the efficacy of hyper-heuristics remains consistent across the same problem instances, even though the re-implemented ones exhibit slightly poor performance. The re-implemented LeanGIHH even outperforms all of the HyFlex hyper-heuristics on the hg4-300-1200 instance, despite it being an easier instance in this domain. In conclusion, I was able to achieve comparable performance in this problem domain. As for possible future work, it would be interesting to explore different re-implementation approaches to determine if better results can be achieved. Currently, the metaheuristics do not suggest a significant out-performance of basic heuristics when utilized by hyper-heuristics, although shortly I will demonstrate that these metaheuristics do not perform well on certain problem domains.
7.3.6 TSP Problem Domain [120 seconds, 8 repeats]

Examining the results of hyper-heuristics re-implemented in SEAGE compared to their original counterparts in the HyFlex framework, the TSP problem domain raises concerns about its correct implementation. The results produced by the metaheuristics in SEAGE are notably poor, with the best performance coming from my DOHH hyper-heuristic. However, these results are still not comparable to the outcomes achieved by the original HyFlex implementation of hyper-heuristics. This performance underscores the dominance of heuristics over metaheuristics in this particular problem domain.

Despite the poor results, the hyper-heuristics in SEAGE demonstrate consistency, with none of them deviating significantly from the results achieved on each instance of this problem domain. This consistent performance indicates a stable behavior of the hyper-heuristics in SEAGE, even if the absolute scores are not competitive with those from the HyFlex framework.

Again, thanks to the heatmap table, it’s evident that the re-implementation did not achieve even a slight improvement in the original performance of the HyFlex hyper-heuristics. This suggests that the metaheuristics cannot even come close to matching the performance of the heuristics. However, many variables could have influenced this result, with the main factor being the poor performance of the metaheuristics themselves, which will be discussed later. In the conclusion, the poor results would suggest the dominance of heuristics over metaheuristics. However, as evident from the previous SAT problem domain and the following FSP, metaheuristics tend to perform closely to the performance of heuristics. This suggests the poor implementation of the TSP problem domain in the SEAGE.

![Table 7.12 HyFlex hyper-heuristics scores on TSP instances 120 seconds, 8 repeats.](image)

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>d1291</th>
<th>pr299</th>
<th>rat575</th>
<th>u2152</th>
<th>usa13509</th>
<th>TSP score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 LeanGIHH</td>
<td>Steven Adriaensen</td>
<td>0.79548</td>
<td>0.99477</td>
<td>0.9652</td>
<td>0.83262</td>
<td>0.84482</td>
<td>0.84628</td>
</tr>
<tr>
<td>2 ISEA</td>
<td>Jiri Kubalik</td>
<td>0.79946</td>
<td>0.99477</td>
<td>0.9611</td>
<td>0.79096</td>
<td>0.81423</td>
<td>0.8182</td>
</tr>
<tr>
<td>3 VNS-TW</td>
<td>Ping-Che Hsiao</td>
<td>0.746</td>
<td>0.90472</td>
<td>0.97852</td>
<td>0.77586</td>
<td>0.75068</td>
<td>0.76491</td>
</tr>
</tbody>
</table>

![Table 7.13 SEAGE hyper-heuristics scores on TSP instances 120 seconds, 8 repeats.](image)

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>d1291</th>
<th>pr299</th>
<th>rat575</th>
<th>u2152</th>
<th>usa13509</th>
<th>TSP score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 DOHH</td>
<td>David Omrai</td>
<td>0.40319</td>
<td>0.60966</td>
<td>0.80968</td>
<td>0.56155</td>
<td>0.72391</td>
<td>0.27892</td>
</tr>
<tr>
<td>2 re-VNSTW</td>
<td>Ping-Che Hsiao / Omrai</td>
<td>0.40843</td>
<td>0.59287</td>
<td>0.46289</td>
<td>0.27921</td>
<td>0.74952</td>
<td>0.27330</td>
</tr>
<tr>
<td>3 re-ISEA</td>
<td>Jiri Kubalik / Omrai</td>
<td>0.42226</td>
<td>0.44442</td>
<td>0.45412</td>
<td>0.26674</td>
<td>0.53586</td>
<td>0.25604</td>
</tr>
<tr>
<td>4 re-LeanGIHH</td>
<td>Misir/Adriaensen/Omrai</td>
<td>0.40317</td>
<td>0.55346</td>
<td>0.64602</td>
<td>0.20004</td>
<td>0.56796</td>
<td>0.29287</td>
</tr>
</tbody>
</table>
7.3.7 FSP Problem Domain [120 seconds, 8 repeats]

Comparing the results of hyper-heuristics re-implemented in SEAGE to their original implementation in the HyFlex framework, it’s noteworthy that the re-implemented hyper-heuristics from SEAGE resemble the results of their original counterparts in HyFlex. They stand only slightly behind, with one instance even outperforming the original implementation. The consistent performance across instances suggests that hyper-heuristics utilizing metaheuristics in SEAGE can achieve results almost as good as those using heuristics in HyFlex.

The best-performing hyper-heuristic from SEAGE in this comparison is now the VNS-TW. This indicates that, despite the overall competitiveness, different hyper-heuristics may excel in specific problem instances or domains when re-implemented in SEAGE.

As mentioned in the SAT results, here, thanks to the heatmap table, we can observe quite similar performance between hyper-heuristics using metaheuristics and heuristics. The original implementation slightly outperforms the re-implementation, yet not by a significant amount. In the HyFlex results, the instances with the best performance are tail100_20_02 and tai500_20_03, both also dominating with re-implemented hyper-heuristics. The only instance of out-performance can be seen on the tai200_20_01, where the re-implemented ISEA hyper-heuristic achieves slightly better performance. In conclusion, we observe similar performance between hyper-heuristics utilizing metaheuristics and those utilizing heuristics. This suggests that metaheuristics can perform as well as heuristics and, with better re-implementation, might even outperform them.

![Table 7.14](image)

Table 7.14 | HyFlex hyper-heuristics scores on FSP instances 120 seconds, 8 repeats.

![Table 7.15](image)

Table 7.15 | SEAGE hyper-heuristics scores on FSP instances 120 seconds, 8 repeats.
The objective of these experiments is to observe the performance of hyper-heuristics within the SEAGE framework, specifically focusing on their ability to manage and utilize metaheuristics. The hypothesis under examination is whether hyper-heuristics, equipped with the capability to learn the performance characteristics of each metaheuristic, can outperform the standalone metaheuristics. The underlying idea is that hyper-heuristics should adeptly schedule, and optimize the use of metaheuristics, leveraging their strengths for enhanced overall performance.

Similar to the previous set of experiments, these experiments use two durations (60 seconds and 120 seconds) for each run of both hyper-heuristics and metaheuristics on four problem domains (FSP, JSP, SAT, and TSP), with 4 and 8 repeats for each duration. This investigates whether doubling the time and repeats leads to improved algorithm performance.

In the final and most extensive experiment, the results will be analyzed on a per-instance basis. This detailed examination tries to provide a deeper performance understanding of each algorithm, across several problem domains.

The ultimate goal is to conclude how hyper-heuristics effectively manage the scheduling of metaheuristics, solutions, and parameters. These findings should contribute to a better understanding of the capabilities of hyper-heuristics within the SEAGE framework, and highlight eventual gaps in their implementation or the SEAGE.
7.4.1 First Experiment [60 seconds, 4 repeats]

In the first experiment, each algorithm ran for 60 seconds, with 4 repeats. My hyper-heuristic, DOHH, results as the top-performing algorithm, demonstrating dominance mainly in the TSP problem domain. In the JSP problem domain, the Tabu Search metaheuristic outperformed all others; in the FSP, the Simulated Annealing outperformed all others, and in SAT, this dominance goes to the VNS-TW hyper-heuristic. The results also highlighted the consistent sub-optimal performance of the TSP problem domain across all algorithms, raising concern about its implementation in the SEAGE.

From the heatmap, we can observe that the performance of hyper-heuristics closely mirrors the performance of the Tabu Search, raising questions about whether the hyper-heuristics are essentially the best-performing metaheuristic with a different name. However, there are a few factors that can influence the performance of hyper-heuristics. Firstly, the current implementation of the hyper-heuristic allocates only 1/10 of the overall time for each metaheuristic, adjusting it based on the performance of the metaheuristics. Additionally, there are poorly performing metaheuristics such as the Ant Colony and Genetic Algorithm, and even Simulated Annealing on the TSP problem domain, providing limited room for improvement. Ultimately, two metaheuristics, Simulated Annealing, and Tabu Search, are responsible for the good results of hyper-heuristics, running with a smaller amount of time. However, these hyper-heuristics were able to achieve similar performance to them. This is just an observation; the most important factor in the end is the quality of results.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>FSP</th>
<th>JSP</th>
<th>SAT</th>
<th>TSP</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 DOHH</td>
<td>David Omrai</td>
<td>0.83437</td>
<td>0.83955</td>
<td>0.81107</td>
<td>0.06329</td>
<td>0.68543</td>
</tr>
<tr>
<td>2 re-VNSTW</td>
<td>Ping-Chu Hsiao / Omrai</td>
<td>0.80993</td>
<td>0.80605</td>
<td>0.86499</td>
<td>0.06003</td>
<td>0.68181</td>
</tr>
<tr>
<td>3 TabuSearch</td>
<td>SEAGE</td>
<td>0.79061</td>
<td>0.8698</td>
<td>0.8045</td>
<td>0.06465</td>
<td>0.67979</td>
</tr>
<tr>
<td>4 re-LeanGILH</td>
<td>Misir/Adriaensen/Omrai</td>
<td>0.80896</td>
<td>0.81859</td>
<td>0.81862</td>
<td>0.04966</td>
<td>0.67296</td>
</tr>
<tr>
<td>5 re-ISEA</td>
<td>Jiří Kubalík / Omrai</td>
<td>0.80547</td>
<td>0.83019</td>
<td>0.78297</td>
<td>0.02197</td>
<td>0.66008</td>
</tr>
<tr>
<td>6 SimulatedAnnealing</td>
<td>SEAGE</td>
<td>0.8402</td>
<td>0.48063</td>
<td>0.67669</td>
<td>0.01804</td>
<td>0.49234</td>
</tr>
<tr>
<td>7 GeneticAlgorithm</td>
<td>SEAGE</td>
<td>0.58789</td>
<td>0.23416</td>
<td>0.19080</td>
<td>0.03204</td>
<td>0.2407</td>
</tr>
<tr>
<td>8 AntColony</td>
<td>SEAGE</td>
<td>0.85775</td>
<td>0.32</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.16 SEAGE algorithms experiment scores 60 seconds, 4 repeats.
7.4.2 Second Experiment [60 seconds, 8 repeats]

In the second experiment, all algorithms ran for the same duration of time (60 seconds), with an extended number of repeats, now 8. We can see, that the hyper-heuristics displayed dominance over other metaheuristics. My hyper-heuristic achieved the highest score, closely followed by VNS-TW in second place and LeanGIHH in third, with ISEA closely behind.

Once again, the results demonstrated proximity to the performance of the best-performing metaheuristic, Tabu Search. Interestingly, the outcome from the Ant Colony and Genetic Algorithm metaheuristics’ results on SAT suggests potential areas for improvement.

From the heatmap, we can observe a similar performance as in the first experiment, where I closely described the problems with it. The performance of hyper-heuristics seems to be more stable across all problem domains. The most notable improvement is on the TSP problem domain, showcasing that where one metaheuristic can’t yield good enough performance, the use of all of them in a hyper-heuristic may. Overall, the performance suggests that the best-performing metaheuristic, Tabu Search, is responsible for most of the quality on each problem domain, with Simulated Annealing helping on the FSP problem domain.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>FSP</th>
<th>JSP</th>
<th>SAT</th>
<th>TSP</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 DOHH</td>
<td>David Omrai</td>
<td>0.83</td>
<td>0.84</td>
<td>0.88</td>
<td>0.25</td>
<td>0.70</td>
</tr>
<tr>
<td>2 re-VNSTW</td>
<td>Ping-Che Hsiao / Omrai</td>
<td>0.81</td>
<td>0.87</td>
<td>0.86</td>
<td>0.53</td>
<td>0.70</td>
</tr>
<tr>
<td>3 re-LeanGIHH</td>
<td>Misir/Adriaenser/Omrai</td>
<td>0.81</td>
<td>0.84</td>
<td>0.88</td>
<td>0.54</td>
<td>0.71</td>
</tr>
<tr>
<td>4 re-ISEA</td>
<td>Jiri Kubalik / Omrai</td>
<td>0.81</td>
<td>0.85</td>
<td>0.76</td>
<td>0.25</td>
<td>0.68</td>
</tr>
<tr>
<td>5 TabuSearch</td>
<td>SEAGE</td>
<td>0.78</td>
<td>0.89</td>
<td>0.78</td>
<td>0.35</td>
<td>0.67</td>
</tr>
<tr>
<td>6 SimulatedAnnealing</td>
<td>SEAGE</td>
<td>0.84</td>
<td>0.89</td>
<td>0.65</td>
<td>0.24</td>
<td>0.50</td>
</tr>
<tr>
<td>7 GeneticAlgorithm</td>
<td>SEAGE</td>
<td>0.57</td>
<td>0.32</td>
<td>0.32</td>
<td>0.60</td>
<td>0.57</td>
</tr>
<tr>
<td>8 AntColony</td>
<td>SEAGE</td>
<td>0.35</td>
<td>0.30</td>
<td>0.10</td>
<td>0.10</td>
<td>0.35</td>
</tr>
</tbody>
</table>

Table 7.17 SEAGE algorithms experiment scores 60 seconds, 8 repeats.
7.4.3 Third Experiment [120 seconds, 4 repeats]

In the third experiment, each algorithm ran for an extended duration of 120 seconds, with 4 repeats. The results indicated slight improvement across all problem domains, and the order of performance remains consistent.

In the JSP problem domain, the DOHH emerged as the dominant performer, with VNS-TW and Tabu Search closely behind. For the SAT problem domain, my DOHH hyper-heuristic takes the first place. In the FSP, all algorithms, except for Genetic Algorithm and Ant Colony, produce results of similar quality.

When comparing the heatmap of the previous experiment with this heatmap, we can see that the most notable change is in the TSP problem domain and the overall score. In this case, all hyper-heuristics resulted in better overall results on each problem domain, yet the poorest metaheuristics resulted in worse performance. A detailed description of what might have affected the results is provided in the first experiment.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>FSP</th>
<th>JSP</th>
<th>SAT</th>
<th>TSP</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOHH</td>
<td>David Omra</td>
<td>0.85554</td>
<td>0.86384</td>
<td>0.88727</td>
<td>0.25807</td>
<td>0.71577</td>
</tr>
<tr>
<td>re-VNSTW</td>
<td>Ping-Che Hsiao / Omra</td>
<td>0.83824</td>
<td>0.86291</td>
<td>0.86252</td>
<td>0.27883</td>
<td>0.70987</td>
</tr>
<tr>
<td>re-LeanGIH</td>
<td>Misir/Adriaensen/Omra</td>
<td>0.84296</td>
<td>0.80521</td>
<td>0.87496</td>
<td>0.25802</td>
<td>0.69529</td>
</tr>
<tr>
<td>re-ISEA</td>
<td>Jiri Kubalik / Omra</td>
<td>0.85293</td>
<td>0.84088</td>
<td>0.80964</td>
<td>0.25800</td>
<td>0.69056</td>
</tr>
<tr>
<td>TabuSearch</td>
<td>SEAGE</td>
<td>0.8211</td>
<td>0.85395</td>
<td>0.80374</td>
<td>0.24198</td>
<td>0.68019</td>
</tr>
<tr>
<td>SimulatedAnnealing</td>
<td>SEAGE</td>
<td>0.85016</td>
<td>0.89723</td>
<td>0.68965</td>
<td>0.02928</td>
<td>0.67997</td>
</tr>
<tr>
<td>GeneticAlgorithm</td>
<td>SEAGE</td>
<td>0.54719</td>
<td>0.34534</td>
<td>0.00202</td>
<td>0.00000</td>
<td>0.24084</td>
</tr>
<tr>
<td>AntColony</td>
<td>SEAGE</td>
<td>0.88474</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.88474</td>
</tr>
</tbody>
</table>

Table 7.18 SEAGE algorithms experiment scores 120 seconds, 4 repeats.
7.4.4 Fourth Experiment [120 seconds, 8 repeats]

In the fourth and final experiment, there was a notable shift in the order of algorithm performance. Tabu Search metaheuristic claimed the top spot, with a slightly superior overall performance compared to hyper-heuristics. Tabu Search demonstrated a slight dominance in the JSP problem domain, with the LeanGIHH hyper-heuristic and DOHH hyper-heuristic closely following. Interestingly, in the TSP problem domain, Tabu Search achieved poorer results than the hyper-heuristics, particularly the DOHH and the VNS-TW. Even though Tabu Search secured the first place, the hyper-heuristics demonstrate a more stable quality across all problem domains, whereas the metaheuristic dominates in only one problem domain to a lesser extent.

The Simulated Annealing metaheuristic excelled in the FSP problem domain, achieving the best performance, with the VNS-TW and ISEA hyper-heuristics closely behind.

This change in algorithm performance order in the fourth experiment adds a perspective to the comparative analysis, showcasing variations in effectiveness across different problem domains.

As I mentioned in the first experiment, from the heatmap, we can see that the performance of hyper-heuristics closely resembles that of Tabu Search. Now, Tabu Search itself outperforms all the hyper-heuristics, though not by a large amount. This raises the question of whether the hyper-heuristics are essentially Tabu Search with a nicer name, adding unnecessary complexity. However, as I described before, there are a few factors that could influence the performance of hyper-heuristics, such as the current implementation, which allocates only a fraction of the overall time, or the implementation of the metaheuristics themselves (Ant Colony resulted in very poor results). This suggests that the performance of the hyper-heuristics is mainly dependent on two metaheuristics, Tabu Search and Simulated Annealing, and that is not much compared to the variety of HyFlex heuristics with different focuses (local search, mutation, etc.). However, these hyper-heuristics were able to achieve similar performance to them and were able to get very close to the performance of the original implementation in HyFlex, not counting the TSP problem domain.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>FSP</th>
<th>JSP</th>
<th>SAT</th>
<th>TSP</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 TabuSearch</td>
<td>SEAGE</td>
<td>0.82294</td>
<td>0.90808</td>
<td>0.89587</td>
<td>0.26027</td>
<td>0.71723</td>
</tr>
<tr>
<td>2 DOHH</td>
<td>David Omrai</td>
<td>0.83737</td>
<td>0.87071</td>
<td>0.88169</td>
<td>0.27805</td>
<td>0.71717</td>
</tr>
<tr>
<td>3 re-LeanGIHH</td>
<td>Misir/Adriaensen/Omrai</td>
<td>0.83835</td>
<td>0.87986</td>
<td>0.8975</td>
<td>0.25628</td>
<td>0.717</td>
</tr>
<tr>
<td>4 re-VNSTW</td>
<td>Ping-Che Hsiao / Omrai</td>
<td>0.85318</td>
<td>0.85018</td>
<td>0.88273</td>
<td>0.27536</td>
<td>0.71486</td>
</tr>
<tr>
<td>5 re-ISEA</td>
<td>Jiri Kubalik / Omrai</td>
<td>0.84967</td>
<td>0.86127</td>
<td>0.87599</td>
<td>0.26014</td>
<td>0.71324</td>
</tr>
<tr>
<td>6 SimulatedAnnealing</td>
<td>SEAGE</td>
<td>0.85441</td>
<td>0.66147</td>
<td>0.72063</td>
<td>0.05815</td>
<td>0.5746</td>
</tr>
<tr>
<td>7 GeneticAlgorithm</td>
<td>SEAGE</td>
<td>0.64949</td>
<td>0.45842</td>
<td>0.26055</td>
<td>0.26055</td>
<td>0.37194</td>
</tr>
<tr>
<td>8 AntColony</td>
<td>SEAGE</td>
<td>0.1326</td>
<td>0.0</td>
<td>0.0</td>
<td>0.26055</td>
<td>0.04809</td>
</tr>
</tbody>
</table>

Table 7.19 SEAGE algorithms experiment scores 120 seconds, 8 repeats.
7.4.5 SAT Problem Domain [120 seconds, 8 repeats]

In a more in-depth analysis of the SEAGE algorithms results on each instance from the SAT problem domain, several noteworthy observations can be made. The performance of the Ant Colony metaheuristic stands out. This is due to poor implementation of logic, which is still under development.

Remarkably, the hyper-heuristics demonstrate a consistent performance across instances, with instances where they outperform metaheuristics, particularly Tabu Search. Despite this, the VNS-TW outperforms or performs the same as the Tabu Search 4/5 times, suggesting a robust and consistent performance across instances.

Among the hyper-heuristics, VNS-TW consistently outperforms my DOHH hyper-heuristic across most instances, except for one case (pg525-2276 instance), where the DOHH hyper-heuristic is outperformed by the ISEA hyper-heuristic. Notably, the overall scores are close to the best-performing metaheuristic, Tabu Search, highlighting the effectiveness and competitiveness of both hyper-heuristics and metaheuristics within the SEAGE framework.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>hg4-300-1200</th>
<th>jar-684-2300</th>
<th>pg-525-2276</th>
<th>pg-525-2336</th>
<th>pg-696-3122</th>
<th>SAT score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 re-LeanGHH</td>
<td>Mistri, Adriensen, Omrani</td>
<td>0.85106</td>
<td>0.91429</td>
<td>0.87179</td>
<td>0.94186</td>
<td>0.88696</td>
<td>0.8975</td>
</tr>
<tr>
<td>2 TabuSearch</td>
<td>SEAGE</td>
<td>0.80851</td>
<td>0.95338</td>
<td>0.88462</td>
<td>0.88372</td>
<td>0.89565</td>
<td>0.89567</td>
</tr>
<tr>
<td>3 re-VNSTW</td>
<td>Ping-Chie Hsieh, Omrani</td>
<td>0.82979</td>
<td>0.95338</td>
<td>0.76923</td>
<td>0.9186</td>
<td>0.89565</td>
<td>0.88273</td>
</tr>
<tr>
<td>4 DOHH</td>
<td>David Omrani</td>
<td>0.82979</td>
<td>0.92381</td>
<td>0.85897</td>
<td>0.88372</td>
<td>0.87826</td>
<td>0.88169</td>
</tr>
<tr>
<td>5 re-ISEA</td>
<td>Jiri Kubalik, Omrani</td>
<td>0.80851</td>
<td>0.92381</td>
<td>0.91026</td>
<td>0.83023</td>
<td>0.7913</td>
<td>0.87599</td>
</tr>
<tr>
<td>6 SimulatedAnnealing</td>
<td>SEAGE</td>
<td>0.7234</td>
<td>0.82857</td>
<td>0.69231</td>
<td>0.67442</td>
<td>0.60957</td>
<td>0.72063</td>
</tr>
<tr>
<td>7 GeneticAlgorithm</td>
<td>SEAGE</td>
<td>0.53191</td>
<td>0.29564</td>
<td>0.34016</td>
<td>0.40994</td>
<td>0.33017</td>
<td>0.32996</td>
</tr>
<tr>
<td>8 AntColony</td>
<td>SEAGE</td>
<td>0.465</td>
<td>0.685</td>
<td>0.475</td>
<td>0.517</td>
<td>0.467</td>
<td>0.472</td>
</tr>
</tbody>
</table>

Table 7.20: SEAGE algorithms scores on SAT instances 120 seconds, 8 repeats.
7.4.6 TSP Problem Domain [120 seconds, 8 repeats]

In a more detailed examination of the algorithms’ results on the TSP problem domain, certain patterns can be visible. Notably, the Ant Colony algorithm consistently has poor results, except for the usa13509 instance, where its performance slightly surpasses that of the greedy algorithm.

The hyper-heuristics demonstrate an overall superior performance compared to the metaheuristics, with an exception in instances like rat525 and pr299. The top-performing algorithm is the DOHH hyper-heuristic, closely followed by VNS-TW and the ISEA hyper-heuristic. The results of the hyper-heuristics closely mirror those of the Tabu Search metaheuristic, with occasional variations where they either slightly outperform or lag. The performance of hyper-heuristics is more stable than that of metaheuristics, suggesting an effective utilization of the strengths of metaheuristics.

From the heatmap table, it is evident that the hyper-heuristics perform better when the metaheuristics they are utilizing perform poorly, adding to the performance through their combination. The results of Tabu Search again resemble the results of hyper-heuristics in almost all instances, with an exception on pr299 where it performs slightly better. Yet, the hyper-heuristics were able to perform more stably across all of the instances. The Ant Colony again shows it couldn’t help in the slightest bit, except on the usa13509 problem instance. The overall algorithms’ performance on this problem domain, when compared with the rest of the problem domains, is very poor, suggesting that the implementation of the TSP problem domain or the metaheuristics on this problem domain is very poor and would need maintenance.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>d1291</th>
<th>pr299</th>
<th>rat575</th>
<th>u2152</th>
<th>usa13509</th>
<th>TSP score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 DOHH</td>
<td>David Omrai</td>
<td>0.45315</td>
<td>0.60966</td>
<td>0.40689</td>
<td>0.20195</td>
<td>0.25365</td>
<td>0.27982</td>
</tr>
<tr>
<td>2 re-VNSTW</td>
<td>Ping-Che Hsiao / Omrai</td>
<td>0.45844</td>
<td>0.59247</td>
<td>0.46281</td>
<td>0.22162</td>
<td>0.24602</td>
<td>0.27328</td>
</tr>
<tr>
<td>3 re-ISEA</td>
<td>Jiri Kubalik / Omrai</td>
<td>0.42228</td>
<td>0.44442</td>
<td>0.45412</td>
<td>0.28673</td>
<td>0.25236</td>
<td>0.25063</td>
</tr>
<tr>
<td>4 re-LeanGHH</td>
<td>Misir/Adriaensen/Omrai</td>
<td>0.45317</td>
<td>0.55346</td>
<td>0.44602</td>
<td>0.28591</td>
<td>0.21835</td>
<td>0.27229</td>
</tr>
<tr>
<td>5 TabuSearch</td>
<td>SEAGE</td>
<td>0.42007</td>
<td>0.69331</td>
<td>0.49781</td>
<td>0.27901</td>
<td>0.20848</td>
<td>0.22205</td>
</tr>
<tr>
<td>6 GeneticAlgorithm</td>
<td>SEAGE</td>
<td>0.41722</td>
<td>0.49386</td>
<td>0.38569</td>
<td>0.20883</td>
<td>0.18375</td>
<td>0.20407</td>
</tr>
<tr>
<td>7 SimulatedAnnealing</td>
<td>SEAGE</td>
<td>0.41311</td>
<td>0.30251</td>
<td>0.40989</td>
<td>0.20883</td>
<td>0.18375</td>
<td>0.20407</td>
</tr>
<tr>
<td>8 AntColony</td>
<td>SEAGE</td>
<td>0.41311</td>
<td>0.30251</td>
<td>0.40989</td>
<td>0.20883</td>
<td>0.18375</td>
<td>0.20407</td>
</tr>
</tbody>
</table>

Table 7.21 SEAGE algorithms scores on TSP instances 120 seconds, 8 repeats.
7.4.7 FSP Problem Domain [120 seconds, 8 repeats]

Overall, the hyper-heuristics demonstrate similar performance as the best metaheuristics, in this case, the Simulated Annealing, which outperformed Tabu Search on this domain, and the Ant Colony metaheuristic consistently produced the poorest results.

Among the hyper-heuristics, VNS-TW emerged as the top-performing algorithm, dominating others except for the tai200_20_01 instance, where ISEA demonstrated slightly better results. Despite this exception, the hyper-heuristics exhibit a close resemblance to the best-performing metaheuristic, the Simulated Annealing.

Hyper-heuristics take on the performance of the best-performing metaheuristics on a given problem domain. This presents a possible strength in solving various problem domains without needing to know how well a certain metaheuristic performs on it. Even with the same overall timespan (having just a fraction of 1/10 of the timespan for each metaheuristic when selected to be run), the hyper-heuristics managed to perform similarly to the best metaheuristic. This illustrates that while certain metaheuristics may exhibit variations in result quality across different problem domains, the hyper-heuristics skillfully employ them to use their strengths where they excel, even though the achieved results may be slightly worse.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>tai100_20_02</th>
<th>tai100_20_04</th>
<th>tai200_20_01</th>
<th>tai500_20_02</th>
<th>tai500_20_03</th>
<th>FSP score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 SimulatedAnnealing</td>
<td>SEAGE</td>
<td>0.85463</td>
<td>0.83601</td>
<td>0.83545</td>
<td>0.94810</td>
<td>0.86687</td>
<td>0.85441</td>
</tr>
<tr>
<td>2 re-VNS-TW</td>
<td>Ping-Che Hsiao / Omrni</td>
<td>0.85442</td>
<td>0.82331</td>
<td>0.83664</td>
<td>0.83705</td>
<td>0.88724</td>
<td>0.85318</td>
</tr>
<tr>
<td>3 re-ISEA</td>
<td>Jiri Kubalik / Omrni</td>
<td>0.83129</td>
<td>0.81402</td>
<td>0.84298</td>
<td>0.84988</td>
<td>0.86293</td>
<td>0.84967</td>
</tr>
<tr>
<td>4 re-LeanGHH</td>
<td>Mitr/Abtaeoen/OMrni</td>
<td>0.83878</td>
<td>0.81402</td>
<td>0.83836</td>
<td>0.82728</td>
<td>0.86099</td>
<td>0.83835</td>
</tr>
<tr>
<td>5 DCHA</td>
<td>David Omrni</td>
<td>0.83197</td>
<td>0.81402</td>
<td>0.841</td>
<td>0.83027</td>
<td>0.84878</td>
<td>0.83737</td>
</tr>
<tr>
<td>6 TabuSearch</td>
<td>SEAGE</td>
<td>0.76707</td>
<td>0.76411</td>
<td>0.81642</td>
<td>0.82058</td>
<td>0.84664</td>
<td>0.82294</td>
</tr>
<tr>
<td>7 GeneticAlgorithm</td>
<td>SEAGE</td>
<td>0.73469</td>
<td>0.68152</td>
<td>0.58565</td>
<td>0.61404</td>
<td>0.68701</td>
<td>0.64949</td>
</tr>
<tr>
<td>8 Ant Colony</td>
<td>SEAGE</td>
<td>0.73469</td>
<td>0.68152</td>
<td>0.58565</td>
<td>0.61404</td>
<td>0.68701</td>
<td>0.64949</td>
</tr>
</tbody>
</table>

Table 7.22 SEAGE algorithms scores on FSP instances 120 seconds, 8 repeats.
7.4.8 JSP Problem Domain [120 seconds, 8 repeats]

With a few exceptions, the DOHH hyper-heuristic in almost all instances claims dominance over the metaheuristics. An exception occurs on the yn1 instance, where the Tabu Search metaheuristic outperforms the hyper-heuristics, with the LeanGIHH hyper-heuristic closely behind. Notably, on the la16 instance, two hyper-heuristics achieve the optimum solution, and others come remarkably close, surpassing the LeanGIHH and the VNS-TW hyper-heuristics.

From the heatmap table, we can see that the overall performance of each hyper-heuristic resembles that of Tabu Search, which overall demonstrates dominance over other algorithms. On the problem instance la16, Tabu Search, DOHH, and ISEA managed to find the optimal solution. On the ft10 problem instance, DOHH resulted in a better result than Tabu Search. This again showcases that hyper-heuristics, though not outperforming the best metaheuristic, manage to utilize the strengths of each one and perform almost as well as the best one. The Ant Colony again shows that its implementation is poorly done and needs close maintenance. Even with the same overall time duration, the hyper-heuristics again show that their strength lies in utilizing the performance of metaheuristics and representing the performance of the best one on the selected problem domain, providing a strong algorithm that can adapt based on the given domain.

<table>
<thead>
<tr>
<th>Hyper-heuristic</th>
<th>Author</th>
<th>ft10</th>
<th>la16</th>
<th>swv01</th>
<th>swv15</th>
<th>yn1</th>
<th>JSP score</th>
</tr>
</thead>
<tbody>
<tr>
<td>TabuSearch</td>
<td>SEAGE</td>
<td>0.95361</td>
<td>0.94346</td>
<td>0.82857</td>
<td>0.90494</td>
<td>0.91739</td>
<td>0.90808</td>
</tr>
<tr>
<td>re-LeanGIHH</td>
<td>Misir/Adriaensen/Omra</td>
<td>0.96937</td>
<td>0.99450</td>
<td>0.86349</td>
<td>0.84691</td>
<td>0.87825</td>
<td>0.87985</td>
</tr>
<tr>
<td>DOHH</td>
<td>David Omrai</td>
<td>0.97423</td>
<td>0.98333</td>
<td>0.87302</td>
<td>0.91111</td>
<td>0.76087</td>
<td>0.87071</td>
</tr>
<tr>
<td>re-ISEA</td>
<td>Jiri Kubalik /Omra</td>
<td>0.93814</td>
<td>0.96667</td>
<td>0.85714</td>
<td>0.88272</td>
<td>0.78261</td>
<td>0.86127</td>
</tr>
<tr>
<td>re-VNSTW</td>
<td>Ping-Ch Hsiao /Omra</td>
<td>0.92268</td>
<td>0.99450</td>
<td>0.86032</td>
<td>0.89506</td>
<td>0.73478</td>
<td>0.85018</td>
</tr>
<tr>
<td>SimulatedAnnealing</td>
<td>SEAGE</td>
<td>0.54124</td>
<td>0.93989</td>
<td>0.4354</td>
<td>0.87778</td>
<td>0.89957</td>
<td>0.66147</td>
</tr>
<tr>
<td>GeneticAlgorithm</td>
<td>SEAGE</td>
<td>0.85567</td>
<td>0.81421</td>
<td>0.54286</td>
<td>0.39556</td>
<td>0.35052</td>
<td>0.45842</td>
</tr>
<tr>
<td>AntColony</td>
<td>SEAGE</td>
<td>0.43561</td>
<td>0.73586</td>
<td>0.54286</td>
<td>0.39556</td>
<td>0.35052</td>
<td>0.45842</td>
</tr>
</tbody>
</table>

Table 7.23 SEAGE algorithms scores on JSP instances 120 seconds, 8 repeats.
In conclusion, this work aimed to evaluate the efficacy of hyper-heuristics re-implementation using more complex metaheuristics within the SEAGE framework compared to simpler low-level heuristics used by hyper-heuristics in the HyFlex framework. HyFlex hyper-heuristics from the CheSC2011 competition served as benchmarks and were re-implemented with minimum changes in the expanded SEAGE framework. The primary objective was to answer whether the re-implementation utilizing metaheuristics would outperform the original implementations. The secondary objective was to implement a new hyper-heuristic to achieve results comparable or superior to those from the CHeSC2011 competition.

Both of these objectives share common prerequisites, which have been implemented during the work on this thesis. To address the need for a more extensive and comprehensive comparison of hyper-heuristics, I’ve extended the number of problem domains to four (TSP, SAT, JSP, and FSP). To assist hyper-heuristics in storing and accessing prior parameters, I’ve implemented a component, Feedback Configurator, that stores information from previous experiments, including the parameters used in metaheuristics, the quality of the found problem solutions, and duration.

To manage solutions and provide easy access for hyper-heuristics, I implemented a Solutions Pool component, eliminating the need for each hyper-heuristic to implement this logic independently. For a better visualization and understanding of the experiment results’ quality, I’ve implemented a visualization method – a heatmap table. This table utilizes colors to distinguish between the best, good, and poor results in both frameworks.

There was also considerable work required to prepare the frameworks for their use in addressing the hypothesis of this work. This involved changing the data type of results (XML to JSON), searching, gathering, and computing metadata used by the Unit metric on the new problem domains (FSP, JSP), debugging and fixing the poor performance of some metaheuristics, and more.

Finally, after preparing all these necessary components and expanding the SEAGE framework, I’ve implemented a new hyper-heuristic, DOHH, to showcase the development of new hyper-heuristics in this environment. Additionally, I re-implemented three top-performing hyper-heuristics from the CHeSC2011 competition (LeanGIHH, VNS-TW, and ISEA), originally implemented in the HyFlex framework.

To fulfill the objectives of this work, I designed a set of experiments to evaluate how performance differs when hyper-heuristics utilize metaheuristics instead of low-level heuristics.

The experiments revealed that, in general, the original implementations in HyFlex achieved better results in the TSP problem domain, and slightly better results in the FSP and SAT problem domains. The re-implemented versions, especially the ISEA hyper-heuristic in the SAT case, performed comparably or even better. However, challenges were observed in the TSP problem domain, suggesting potential issues with the SEAGE implementation.
When comparing the re-implemented hyper-heuristics with the metaheuristics in the SEAGE, the best metaheuristics usually performed just slightly worse (with more unstable solutions quality across problem domains). These similar performances may be possibly influenced by the current implementation of hyper-heuristics, which allocates only a $1/10$ of the overall time for each metaheuristic run, adjusting it based on their performance. Overall hyper-heuristics seem to take on the performance of the best-performing metaheuristics on a given problem domain. This presents a possible strength in solving various problem domains without needing to know how well a certain metaheuristic performs on it.

Even with the same overall timespan (with the fraction of time for each metaheuristic at a time), the hyper-heuristics always performed similarly to the best metaheuristics. This suggests that despite the dominance of certain metaheuristics in specific problem domains, the hyper-heuristics make use of their strengths where they perform well, even if it results in slightly lower performance at times.

Regardless, thanks to this effort, implementing new hyper-heuristics in the SEAGE framework has become more straightforward. The framework’s database enhances the potential for in-depth studies on the performance of each implemented problem domain. I demonstrated these capabilities by implementing my new hyper-heuristic, DOHH, which proved to be on par with CHeSC2011 hyper-heuristics, both in their original form (on SAT and FSP problem domains) and when re-implemented in the SEAGE framework (across all problem domains).

It is worth exploring whether implementing low-level heuristics in the SEAGE framework could further enhance hyper-heuristics performance. This consideration comes from the time complexity required for a good solution with metaheuristics, being more complex than heuristics.

While the hypothesis couldn’t be rejected nor accepted this work came with a significant outcome, I was still able to implement several new functional hyper-heuristics in the SEAGE framework, which may serve as a starting point for the next possible studies in this field.
Bibliography


Contents of the Attachment

The project is also available at [https://github.com/seage](https://github.com/seage)

readme.txt..........................a brief description of the attachment content.
exe
  _seage ......................directory with the executable SEAGE framework.
  _hyflex ....................directory with the executable HyFlex framework.
src
  _impl
    _seage ..................the SEAGE source code made during this master’s thesis
    _hyflex ..................the HyFlex source code made during this master’s thesis
    _other .....................other source code made during this master’s thesis
  _thesis .......................source code of master’s thesis in \LaTeX

thesis
  _thesis.pdf.............................master’s thesis in PDF
