FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# A New Hyperheuristic Algorithm for Cross-Domain Search Problems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Computational Intelligence**

eingereicht von

**Andreas Lehrbaum**
Matrikelnummer 0205277

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Priv.-Doz. Dr. Nysret Musliu

Wien, 30.11.2011

(Unterschrift Verfasser) (Unterschrift Betreuung)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# A New Hyperheuristic Algorithm for Cross-Domain Search Problems

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Computational Intelligence**

by

**Andreas Lehrbaum**
Registration Number 0205277

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Priv.-Doz. Dr. Nysret Musliu

Vienna, 30.11.2011

(Signature of Author) (Signature of Advisor)

# Erklärung zur Verfassung der Arbeit

Andreas Lehrbaum
Heiligenstädter Straße 115/2/44, 1190 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

# Acknowledgements

I would like to express my deep gratitude to my advisor, Priv.-Doz. Dr. Nysret Musliu, who guided me with great patience and encouragement throughout the process of this work. Without him, I would not have developed my keen interest in the area of AI problem solving and without his competent guidance, I could never have finished this work. I am also very grateful for the care and support from my loving parents who made it possible for me to enjoy an academic education. Finally, I would like to thank my partner, for she bore with me in difficult times, gave me strength through her love and never lost faith in me.

# Abstract

Hyperheuristics are an emergent area of search methodologies which try to address computationally hard problems at a new level of abstraction. Instead of having a single algorithm that is optimised to perform well on a certain class of problem instances, hyperheuristics try to leverage the power of a whole set of problem specific heuristic algorithms. By combining and parametrising these heuristics or heuristic components in different ways, the overall algorithm should be able to perform better over a wider range of problem instances. Ideally, a hyperheuristic does not need to know anything about the problem class it operates on and only very little about the available heuristics. Because of this, hyperheuristics can often be applied without modifications to whole new problem domains and therefore represent a further step towards a very general problem solving algorithm.

This thesis describes a new hyperheuristic algorithm that was specifically designed to be completely problem domain independent and which works at a clearly defined level of abstraction. The algorithm is structured in different search phases that try to balance intensification and diversification of the search process. The available low-level heuristics are evaluated repeatedly in terms of certain properties and ranked according to a quality based metric that was found to work well across all tested low-level heuristics. Furthermore, the algorithm incorporates ideas from a number of different areas of metaheuristic research, such as iterated local search, simulated annealing, tabu search and genetic algorithms.

The main goal of this work was to develop a hyperheuristic that achieves good overall performance on a wide variety of unrelated search domains. In order to assess the generality of the algorithm, it was tested on six different, well-studied problem domains from the field of combinatorial optimisation. A version of the algorithm was also submitted as a candidate for the Cross-domain Heuristic Search Challenge 2011 organised by the University of Nottingham. Out of the twenty participants from all over the world, our algorithm was ranked sixth in the final competition.

# Kurzfassung

Hyperheuristiken bilden ein aufstrebendes Forschungsgebiet für Suchstrategien, welche die Einführung einer neuen Abstraktionsebene nützen um komputational schwierigen Problemen zu begegnen. Anstatt einen einzelnen Algorithmus zu verwenden, der auf eine bestimmte Klasse von Probleminstanzen optimiert ist, versuchen Hyperheuristiken die individuelle Leistungsfähigkeit einer gegebenen Menge von problemspezifischen Heuristiken auszunutzen. Durch die unterschiedliche Kombination und Parametrisierung dieser Heuristiken sind Hyperheuristiken theoretisch dazu in der Lage, gute Ergebnisse über einen größeren Bereich von Probleminstanzen zu erzielen. Idealerweise operieren Hyperheuristiken ohne jegliches Wissen bezüglich der zu lösenden Problemklasse und nur mit sehr geringem Wissen über die zur Verfügung stehenden untergeordneten Heuristiken. Aus diesem Grund können Hyperheuristiken auch oft ohne große Modifikationen auf neue Problemklassen angewendet werden. Damit stellen sie einen weiteren Schritt auf dem Weg zu einem universellen Problemlösungsalgorithmus dar.

Die vorliegende Arbeit beschreibt eine neue Hyperheuristik, die speziell darauf ausgelegt wurde, gänzlich unabhängig von einer bestimmten Problemklasse zu arbeiten und die sich dafür einer sehr klaren Abstraktionsebene bedient. Der Algorithmus ist in eine Anzahl unterschiedlicher Suchphasen untergliedert, welche für ein geeignetes Verhältnis zwischen der Intensivierung und der Diversifizierung des Suchprozesses sorgen. Die verfügbaren untergeordneten Heuristiken werden fortlaufend anhand mehrerer Eigenschaften bewertet und mit einer neuen qualitätsbasierten Metrik gereiht. Weiters vereinigt der Algorithmus eine Reihe von Ansätzen aus verschiedenen Gebieten der Forschung an Metaheuristiken, namentlich Iterated Local Search, Simulated Annealing, Tabusuche und genetischen Algorithmen.

Das Hauptziel dieser Arbeit war die Entwicklung einer Hyperheuristik die gute universelle Leistung über eine Vielzahl unterschiedlicher Problemklassen aufweist. Um die allgemeinen Problemlösungseigenschaften des Algorithmus bewerten zu können, wurde er anhand sechs verschiedener populärer Problemklassen aus dem Bereich der kombinatorischen Optimierung getestet. Eine Variante des Algorithmus wurde als Kandidat bei der Cross-domain Heuristic Search Challenge 2011 von der University of Nottingham eingereicht. Unter den 20 teilnehmenden Kandidaten aus aller Welt belegte unser Algorithmus im finalen Bewerb den sechsten Platz.

# Contents

**List of Tables** x

**List of Figures** xi

**1 Introduction** 1
1.1 Motivation . . . 1
1.2 Aim of this Thesis . . . 3
1.3 Results . . . 3
1.4 Organisation . . . 4

**2 Hyperheuristics** 5
2.1 History . . . 5
2.2 Classification of Hyperheuristics . . . 7
Heuristic Selection . . . 9
Heuristic Generation . . . 9
Online Learning Hyperheuristics . . . 9
Offline Learning Hyperheuristics . . . 10
Non-Learning Hyperheuristics . . . 10
2.3 State of the Art . . . 10
2.4 Towards Cross-domain Hyperheuristics . . . 11

**3 Problem Definition** 13
3.1 Goal Description . . . 13
3.2 Problem Domains . . . 14
Max-SAT . . . 14
Bin-Packing . . . 15
Permutation Flow-Shop . . . 16
Personnel Scheduling . . . 18
Travelling Salesman Problem . . . 19
Vehicle Routing Problem . . . 21
3.3 Evaluation Method . . . 22
Test Data . . . 23

Framework Description . . . . . . . . . . 23
Scoring System . . . . . . . . . . 27

**4 Algorithm Description** **29**
4.1 Overview . . . . . . . . . . 29
4.2 Detailed Description of the Algorithm . . . . . . . . . . 30
Initialisation Phase . . . . . . . . . . 30
Serial Search Phase . . . . . . . . . . 30
Quality Updates . . . . . . . . . . 31
Generation of Mutated Solutions . . . . . . . . . . 31
Parallel Search Phase . . . . . . . . . . 32
Working Solution Selection . . . . . . . . . . 32
Additional Mechanisms . . . . . . . . . . 32
Constant Assignments . . . . . . . . . . 33
4.3 Pseudocode Implementation . . . . . . . . . . 33

**5 Experimental Results** **37**
5.1 Discussion of Results . . . . . . . . . . 39
5.2 Description of Competing Algorithms . . . . . . . . . . 41
AdapHH . . . . . . . . . . 42
VNS-TW . . . . . . . . . . 45
ML . . . . . . . . . . 47
Pearl Hunter . . . . . . . . . . 49
Additional Approaches . . . . . . . . . . 49

**6 Conclusion** **51**
6.1 Further Research . . . . . . . . . . 51

**Bibliography** **53**

# List of Tables

3.1 Benchmark results of 30 consecutive runs. . . . . . . . . . . 23
3.2 Max-SAT instances . . . . . . . . . . 24
3.3 Bin-packing instances . . . . . . . . . . 24
3.4 Personnel scheduling instances . . . . . . . . . . 25

3.5 Pemutation flow-shop instances . . . 25
3.6 Travelling salesman problem instances . . . 26
3.7 Vehicle routing problem instances . . . 26

5.1 Max-SAT results . . . 39
5.2 Bin-packing results . . . 40
5.3 Personnel scheduling results . . . 42
5.4 Permutation flow-shop results . . . 43
5.5 Travelling salesman problem results . . . 45
5.6 Vehicle routing problem results . . . 46
5.7 Final competition ranking . . . 47

# List of Figures

2.1 Metaheuristic classification . . . 8
2.2 Hyperheuristic classification . . . 8
2.3 Timeline of hyperheuristics developments . . . 9

3.1 HyFlex abstraction model for hyperheuristics . . . 27

4.1 Execution cycle of the algorithm . . . 30
4.2 Internal structure of solution array . . . 33

5.1 Max-SAT results . . . 38
5.2 Bin-packing results . . . 41
5.3 Personnel scheduling results . . . 41
5.4 Permutation flow-shop results . . . 44
5.5 Travelling salesman problem results . . . 44
5.6 Vehicle routing problem results . . . 47
5.7 Total results . . . 48

CHAPTER 1

# Introduction

## 1.1 Motivation

It is a long-standing goal of the artificial intelligence research community to design an algorithm that is able to solve problems from a wide range of different problem classes. The first attempts can be dated back to the very beginnings of artificial intelligence research in the late 1950s. One of the earliest representatives was the General Problem Solver[77], which used an abstract representation of states and rules to transform one problem state into another. A means-end analysis was employed to gradually get closer to the desired goal state. Although the required descriptions needed to be so detailed that it could only be applied to toy problems such as the "Tower of Hanoi"[64], it was already specifically designed to be able to tackle almost any problem class and it therefore shows the great ambitions of early AI research.

As soon as researchers try to address problems of the real-world instead of just toy problems, it becomes evident that these are often inherently hard. Complexity research showed, that many real-world problems fall in the complexity class of NP-hard problems[21]. This means that there exists no algorithm that in general is able to compute an optimal solution within a timespan that is polynomial in the size of the given problem instance (unless $P = NP$). In other words, to optimally solve such problems there is no fundamentally faster way than iterating over all possible solutions and check for the best. Even the ever-increasing amount of processing capacity available with modern computer technology can often not match the computational complexity of NP-hard problems. A good practical example is the problem of vehicle routing[31]. Countless delivery companies in the world face the problem of planning an optimal route for their fleet of delivery vehicles in order to service all their customers in the shortest amount of time. A problem of great practical importance, yet the largest non-trivial problem instance that could be solved optimally to date,

has only 135 customers[7] - and that is for a greatly simplified version of the actual real-world problem.

Because of this rather unsatisfactory situation, researchers began to develop efficient heuristic methods to deal with large real-world problem instances. Heuristics often provide good solutions to hard problems in a short amount of time, but they cannot give any guarantee that they will find the optimal solution. Some heuristics do however provide guarantees regarding the approximation ratio of an optimal solution. For example, as shown in [20], the Christofides heuristic for solving the travelling salesman problem is able to always produce solution tours that are no longer than 150% of the shortest possible tour. Besides often not giving optimal solutions, heuristics have another big disadvantage: they are very problem specific. Most heuristics only work for a clearly defined set of problems and there is no easy way to devise a good heuristic for a given new problem domain.

In order to address this problem, researchers began to think about general heuristics that can be applied to any problem domain. Since almost all problem solving algorithms can be formulated as some kind of search for an optimal solution quality within a certain space of solutions, many different search strategies were developed in order to perform such searches efficiently. This work lead to the formation of a new field of research, called ”metaheuristics” (first mention of the term in [46]).

Metaheuristics attempt to provide a non problem-specific optimisation algorithm which explores a search space in a guided manner in order to quickly find (near-) optimal solutions (see [9] for a list of accepted definitions for metaheuristics). The great hope was, to find one type of algorithm that performs an efficient search, no matter how the solution space is structured. Researchers pursued a wide variety of different approaches and often found natural processes to be very useful prototypes for new optimisation algorithms. This lead to the development of algorithms like Simulated Annealing[63] (modelling a physical cooling process), Ant Colony Optimisation[33] (mimicking the way a collection of ants finds a short way to a food source), Genetic Algorithms[52] (implementing the biological process of gene mutation and recombination) or Particle Swarm Optimisation[61] (imitating the movements of a bird flock). Very successful metaheuristics that were not inspired by nature are Iterated Local Search[69] (escape local optima by restarting local search with a perturbed solution) and Tabu Search[47] (temporarily block visited solutions from being re-visited during the search). Researchers began to refine their favourite approaches and often tried to demonstrate their superiority over competing algorithmic paradigms.

The hope to find the one universally predominant class of metaheuristics had to be finally given up after theoretical results like the ”No free lunch” theorems[109] were developed. They define a fundamental limit for the efficiency of any given search strategy when averaged over all possible problem classes. It was basically found out, that the average search space has very little structure to be exploited and that no matter how an algorithm operates, it is possible to come up with problem instances at which

the algorithm performs poorly.

One hope still left is that an algorithmic approach can be developed that might still dominate all the others on only all "interesting" problem instances. Since an integral part of the no free lunch theorems is that they are only valid when taking into account absolutely all possible problem spaces, it cannot be ruled out that the subset of all practical real-world problem instances is indeed dominated by a single algorithmic approach. It is to be expected that no one simple metaheuristic can meet this requirement but that it will have to incorporate (or gather) some kind of knowledge about the problem class it is working on in order to improve the average performance within this domain.

Taking advantage of such domain knowledge to improve the quality of a general algorithm was tried in many different ways. An interesting approach to this task was introduced in [25], which describes the concept of a hyperheuristic, acting on a higher level than traditional metaheuristics. According to [15]: "A hyperheuristic is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems". By utilising a set of existing (low-level) heuristics or heuristic components, they try to solve problems more efficiently or robustly than these individual low-level components could by themselves. Generalising the original idea of hyperheuristics to adapt to a specific problem instance within one problem class, it should also be possible to adapt to completely different problem classes altogether. Hyperheuristics are therefore another step towards the decades-old dream of finding a single, universally applicable problem solving technique for all practical problems.

## 1.2 Aim of this Thesis

This thesis pursues the following goals:

1. to present the concept of hyperheuristics and discuss its application to cross-domain search problems
2. to develop a new hyperheuristic algorithm that performs well over a range of different problem domains and implement it on top of an existing hyperheuristic framework
3. to experimentally evaluate the characteristics of the proposed algorithm on a set of well-known benchmark datasets

## 1.3 Results

The main results of this thesis are:

1. we developed a new hyperheuristic algorithm that introduces a novel way of switching between different search phases, uses an adaptive ranking mechanism

and a quality based selection strategy to take best advantage of the available low-level heuristics

2. we provide extensive test-results and performance characteristics of the algorithm on the following problem domains:
   - Max-SAT
   - One-dimensional bin-packing
   - Personnel scheduling
   - Permutation flow-shop
   - Vehicle routing problem
   - Travelling salesman problem
3. results show that our algorithm achieves a good overall problem-solving capability and performs especially well with the Max-SAT and personnel scheduling instances
4. we implemented our algorithm on top of thy HyFlex framework [10] and submitted it as a candidate for the Cross-domain Heuristic Search Challenge 2011 [1] by the University of Nottingham where we were ranked 6th out of 20 participants

## 1.4 Organisation

The remaining parts of this thesis are organised as follows: in Chapter 2, we give an introduction to hyperheuristics, beginning with the historical developments and a classification scheme for current hyperheuristics. We continue to briefly describe some of the notable state of the art hyperheuristic techniques and give an overview of research towards cross-domain hyperheuristics. In Chapter 3 we define the problem classes used for testing of the developed hyperheuristic, describe which low-level heuristics were used for our experiments and also cover existing hyperheuristic approaches for each of the individual problem domains. We also give a description of the hyperheuristic framework used for the development of the algorithm and explain the evaluation method. In Chapter 4, we then introduce our newly developed algorithm by giving a detailed description of the separate components as well as a pseudocode implementation for the main functional entities. Chapter 5 discusses the results and compares the algorithm to a number of competing high quality cross-domain hyperheuristics. Chapter 6 concludes this thesis with a summary of the results and contributions of this work. It also gives an insight into currently ongoing development work as well as an outlook on promising directions of future research regarding this algorithm.

[1] `http://www.asap.cs.nott.ac.uk/chesc2011`

CHAPTER 2

# Hyperheuristics

Hyperheuristics are a way to incorporate existing problem-class specific domain knowledge in the form of simple low-level heuristics into a higher-level search strategy which schedules and guides the execution of the lower-level heuristics. The idea is that an ensemble of heuristics, orchestrated by a top-level strategy, is able to perform better on average at solving a wide range of problems as any of the underlying heuristics alone. Another common use-case for hyperheuristics is to quickly address new or uncommon problem types where not much domain knowledge is available. In order to apply a hyperheuristic to such a new problem domain it is sufficient to define very simple heuristics which often follow directly from the solution representation of the new problem (eg. a simple hill-climbing local search using small changes in the solution vector as search steps). In many cases, the flexible and robust nature of the top level hyperheuristic can produce good results without spending much effort on analyzing and manually adapting the algorithm to a new domain.

This black-box approach when it comes to solving a problem is similar to the principle of metaheuristics. In fact, it could be argued, that hyperheuristics are just a somewhat more complex class of metaheuristics. There is however an important conceptual distinction between the two: metaheuristics search within a space of candidate solutions to a problem, whereas hyperheuristics always search within a space of heuristics to solve the problem. As such, hyperheuristics introduce a new level of abstraction and can be seen as a natural extension of the metaheuristic concept.

## 2.1 History

Although the name "hyperheuristic" itself was only coined rather recently, the underlying ideas behind hyperheuristics are much older. Methods to combine different scheduling rules in factory production were already described in the early 1960s by

Fisher and Thompson in [37]. In their far-sighted work, they concluded that probabilistic learning could be used to find a suitable sequence of scheduling rules which would improve upon their respective individual performance. This is especially noteworthy because at that time, not even the idea of metaheuristics existed and that only very rudimentary local search techniques were available.

Probably due to the generally immature state of the computational search methodologies and the limited amount of available computing power, nobody really picked up the idea until the early 1990s. Some concepts similar to hyperheuristics were however already investigated before that. In 1973, Rechenberg proposed a method of adaptively changing the mutation rate of evolutionary algorithms (described in [88]) and in 1976, Rice formulated the algorithmic selection problem in [91]. The question of which algorithm is likely to best solve a given problem was examined and lead to the field of meta-learning.

In 1993, the term ”hyperheuristic” was still not in use, but the exact same concept was used in [49], where a hill-climbing algorithm operates on a search space of control strategies for satellite communication. The problem is known to be NP-hard and there were real-life problem instances that could not be solved in a reasonable amount of time. Using the newly proposed algorithm they could however increase the number of solvable instances as well as shorten the time to find a solution when compared with existing techniques which in part relied on human experts. This result was one of the first that showed the practical viability of a hyperheuristic approach and paved the way for further research. The work was done only shortly after, for the first time, sequences of heuristics were formally defined as a search space and suitable neighbourhood structures were given in [98].

Then in 1995, one of the first heuristic generation methods based on genetic algorithms was introduced in [107], which a few years later led to the development of the TEACHER system (”Techniques for the Automated Creation of HEuRistics”) in [106]. Teacher was also one of the first systems that was successfully applied to a whole range of different problem domains (circuit routing, load balancing, process mapping, etc.). It worked by addressing the decomposition of an existing problem solver for a given domain into smaller heuristic methods and the classification of an application domain into subdomains. Statistical evaluation was performed on the individual subdomains in order to find out which heuristic methods worked best with each. Finally, Teacher tried to generate new improved heuristic methods and find ones that worked well across the whole problem domain.

Another very similar approach to hyperheuristics was introduced in 1998 with the Squeaky Wheel optimisation method as described in [58]. Squeaky Wheel uses a dual search space: one is the normal space of solutions to the given problem instance, the other is a priority space of solution components. Solutions are constructed by a greedy algorithm that uses a priority ordering of the solution parts as input. Whenever changing a part of a solution is identified as likely to improve the quality of the overall

solution, its priority is increased and therefore made more likely to be improved upon by the constructive heuristic. By simultaneously operating in two different search spaces, Squeaky Wheel can effectively escape local minima in the search process and proved to produce encouraging results in scheduling and graph colouring domains.

The term "hyperheuristic" itself dates back to the year 1997 - it was first used to describe a sequence of several artificial intelligence methods and their parameters within a distributed automated theorem proving system (ATP)[32]. These sequences were used to facilitate so-called reproduction runs, which made it possible to reproduce the exact steps of a learning process that led to a particular solution.

Three years later in the year 2000, the term was used independently and with a slightly different meaning, to describe "heuristics to choose heuristics"[25]. In this work, the hyperheuristic model with the clear distinction between a problem independent top-level heuristic selection process and the problem specific low-level heuristics was introduced. Three different selection processes for the underlying low-level heuristics were examined: a random approach, a greedy approach and a choice function-based approach. The random approach simply chooses one of the available heuristics randomly at each step. The greedy approach evaluates the changes in the objective function value with each of the available heuristics and chooses the best for each step. The choice function estimates how effective each of the low-level heuristics will be, based on the current state of the search space and the knowledge gained from previous applications of the heuristics. Four different choice functions were suggested and compared. This newly introduced hyperheuristic approach has been successfully applied to a scheduling task and inspired much further research in the area of hyperheuristics.

A timeline of some of the important developments in the field of hyperheuristics is shown in figure 2.3.

## 2.2 Classification of Hyperheuristics

Albeit being a rather young area of research, hyperheuristics already span a wide range of different approaches and techniques. This comes to no surprise, as hyperheuristics can build on the very established foundations of computational search processes and metaheuristic research. In fact, pretty much every major metaheuristic approach has also been applied at the hyperheuristic level. Because of this, it is often also possible to identify groups of hyperheuristics using traditional metaheuristic categories. Figure 2.1 shows an attempt of classifying a number of important metaheuristic concepts according to their main characteristics.

When looking purely at the hyperheuristic level, the following main categories can be identified (according to [19]):

- Heuristic selection methodologies

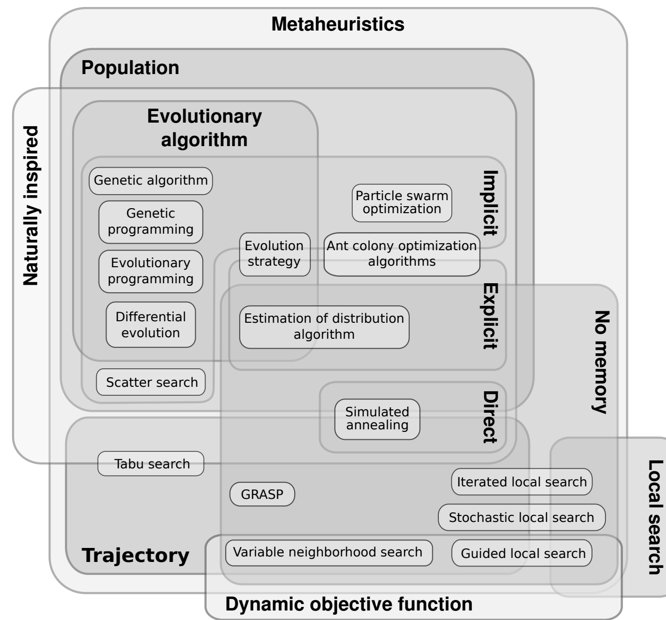


**Figure 2.1:** Some classes of metaheuristics (based on [35])

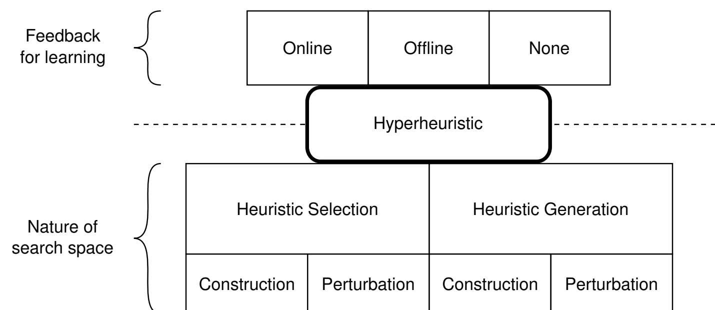


**Figure 2.2:** Hyperheuristic classification (based on [19])

- Heuristic generation methodologies
- Online learning hyperheuristics
- Offline learning hyperheuristics
- Non-learning hyperheuristics

The first two address the nature of the search space used by the hyperheuristic, while the remaining ones concern the source of the feedback for the learning process. An illustration of this classification scheme is depicted in figure 2.2.

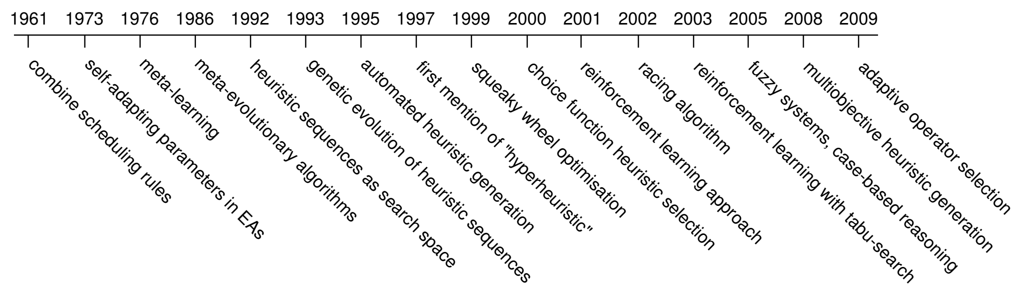


**Figure 2.3:** Timeline of hyperheuristics developments

### Heuristic Selection

Heuristic selection deals with choosing a suitable sequence of low-level heuristics to apply from a pool of available heuristics. The search space for this type of hyperheuristic is therefore the set of all possible sequences of low-level heuristics. Pure heuristic selection processes do not modify the underlying heuristics, but only schedule them. Depending on the actual implementation, it is however often possible to parametrise them. Because of this, a clear distinction between pure heuristic selection and heuristic generation can not always be made.

Notable representatives of this class can be found in [37, 85, 101, 4].

### Heuristic Generation

Heuristic generation also allows for the creation of new low-level heuristics. This usually works by exchanging certain parts of problem-specific heuristics or by applying genetic programming techniques (see [65]). Some of the approaches could strictly also be viewed as heuristic selection, albeit working partly on a lower level of heuristic components. Notable representatives of hyperheuristics employing heuristic generation can be found in [1, 100, 79, 87].

### Online Learning Hyperheuristics

Online learning hyperheuristics usually do not have any prior knowledge about the problem class they are working on. They have to gain knowledge about the available low-level heuristics as well as the problem instance they are working on purely by trial and error. Common techniques for online learning come from the field of reinforcement learning, which tries to continuously incorporate knowledge from the progress of the search itself in order to perform more informed actions in the future.

### Offline Learning Hyperheuristics

The idea with offline learning is that a set of training problems is presented to the hyperheuristic before it is actually used. The hyperheuristic then tries to learn some form of knowledge from these training samples which hopefully can be applied successfully to problems outside the training dataset as well. The extracted knowledge can be in various forms: from rules or classifiers (as in [70] and [94]) over case-based reasoning (see [18]), to genetically evolved structures (for example [93] and [102]).

### Non-Learning Hyperheuristics

Hyperheuristics that do not learn were mainly used in the initial work on hyperheuristics by Cowling et al.[25]. While they are clearly inferior to hyperheuristic selection processes that apply some learning mechanism, they can be useful as a baseline comparison when working with hyperheuristics.

## 2.3 State of the Art

Soon after the re-introduction of the hyperheuristic concept by Cowling et al. in 2000[25], many different approaches were examined using the newly proposed model. One of the first improvements on the choice function based approach by Cowling et al. was the use of reinforcement learning (as described in [26]). Reinforcement learning uses a generic model of an *environment* that is in a certain *state* and can be influenced by performing *actions* which are chosen by a certain *policy*. The notion of a *reward* is used to increase or decrease the score of *state,action*-pairs that have been previously encountered. See [59] for a survey on this technique.

When it comes to hyperheuristics, *actions* are usually the application of the low-level heuristic and the *reward* is derived from the change in the solution quality. A notable reinforcement learning based hyperheuristic is described in [75] and a combination with a tabu-search mechanism in [17]. With simulated annealing, another common metaheuristic technique is successfully added to the reinforcement learning tabu search in [34].

There is no absolute requirement for hyperheuristics that the top-level process has to be a heuristic itself. Several knowledge based strategies have been successfully applied as heuristic selection processes as well. In [18], a case-based reasoning process is employed to create a hyperheuristic that produces very good results in a timetabling problem. Another comparative work with a knowledge based classifier system for hyperheuristic selection was done in [103].

Recent interesting developments in the area of hyperheuristics include multi-objective heuristic generation by a genetic algorithm as described in [100], where a genetic programming approach is used to solve a variant of the job-shop problem that has to be optimised simultaneously for minimal makespan, minimal mean tardiness and mini-

mal mean flow time. This work expands the area of hyperheuristics to the much more challenging problem domains of multi-objective optimisation.

Another promising area of hyperheuristic research is currently being explored with multi-agent systems as described in the work of [83]. An agent-based cooperative hyper-heuristic framework is proposed, that includes a number of low-level heuristic agents and a single cooperative hyper-heuristic agent. The heuristic agents are allowed to communicate synchronously or asynchronously through the hyperheuristic agent, which tries to balance their individual strengths and weaknesses. It has been shown that this approach can yield superior performance compared to existing sequential hyper-heuristics at a set of permutation flow-shop benchmark instances.

A good survey of currently existing hyperheuristic techniques can be found in [15].

## 2.4 Towards Cross-domain Hyperheuristics

The majority of hyperheuristic research has been devoted to single domain applications. The goal was mostly to make existing metaheuristics more robust and to adapt to a wider range of problem instances within one domain. Notable exceptions are [81] and [79] which use multi expression programming and linear genetic programming to evolve other evolutionary algorithms. This approach was applied to the travelling salesman problem, the quadratic assignment problem and general function optimization problems. In [42], a source-code generator is described, which produces C++ code via automatic synthesis of stochastic local search algorithms. This system can generally be applied to many different problem domains as well. The problem description has to be provided in an XML syntax and the resulting code has to be completed manually with problem specific methods.

Every multi-disciplinary problem solving algorithm faces the question of how to choose a generic problem representation. In addition to that, hyperheuristics need a unified interface to access multiple different low-level heuristics, regardless of the problem domain they are designed to work with. The HyFlex framework as described in [12] was developed to directly address these issues. Building on the research results from [84] and [8], it provides probably the most abstract and generic interface for hyperheuristic algorithms to date. HyFlex comes with a rather large set of comparative problem domains and test instances, shifting away the focus from improving existing solution strategies within a problem domain, towards a more generally applicable problem solving ability.

In [11], a brief comparative study on the performance of different state of the art hyperheuristics using the HyFlex framework is conducted. A number of basic algorithms from the literature was implemented and tested on three different problem domains (bin-packing, permutation flow-shop and personnel scheduling). Interestingly, a simple version of *iterated local search* was found to produce the best overall results.

In 2011, an international competition explicitly targeted at cross-domain hyperheuristics was organised by the University of Nottingham, in order to encourage research in that direction and to objectively compare different hyperheuristic approaches in that area. The "Cross-domain Heuristic Search Challenge 2011" strongly built upon the HyFlex framework and was the first competition of its kind. This tournament was especially important, because objectively comparing cross-domain hyperheuristics is a very difficult task and probably one of the reasons why not very much research has been done in that direction. A common framework like HyFlex not only makes sure that everybody uses the same implementations of low-level heuristics, but also that the information available to the hyperheuristic is exactly the same for each of the candidates.

The results of this competition are certainly an important contribution towards a more comprehensive understanding of problem independent top-level solution strategies and a major step forward in cross-domain hyperheuristic research.

CHAPTER 3

# Problem Definition

In this Chapter, we will define the problem environment which we used for the work on our hyperheuristic. We will introduce the different problem classes used as application domains for the developed algorithm as well as the available low-level heuristics and the specific benchmark instances for training and testing that were provided with the HyFlex framework. In addition to that, we will also shortly address existing hyperheuristic research works that deal with each of the problem domains. At the end of the Chapter we will describe the underlying HyFlex framework that was used for the implementation of our algorithm as well as the evaluation method employed.

## 3.1 Goal Description

The main goal of this research was the development of a hyperheuristic algorithm that is universally applicable, regardless of the specific problem class. It is to be expected, that increased generality and robustness of an algorithm come at the cost of sub-optimal individual results. In fact, for every particular instance there exist more specialised heuristics that produce better results. Previous hyperheuristic research often focused on a single problem domain and produced very competitive or even new best known solutions for individual instances (see for example [86]). This work however does not strive for optimal performance on individual instances or even problem classes. It is therefore somewhat distinct from the majority of previous works on hyperheuristics.

In order to analyse the performance of the algorithm, the following problem domains were used for our experiments:

1. Maximum satisfiability (Max-SAT)
2. One-dimensional bin-packing

3. Permutation flow-shop
4. Personnel scheduling
5. Travelling salesman problem (TSP)
6. Vehicle routing problem (VRP)

## 3.2 Problem Domains

### Max-SAT

The well-known boolean maximum satisfiability problem is to find an assignment to all boolean variables of a set of logical clauses, that leads to the highest number of these clauses evaluating to true. An example of a problem instance in conjunctive normal form is given in formula 3.1.

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \tag{3.1}$$

In this case, a possible solution would be the assignment $x_1 = true$, $x_2 = false$, $x_3 = true$, $x_4 = true$, which would satisfy all three clauses. The problem is known to be NP-hard[39] and many other problems can easily be expressed as an instance of a SAT problem. In fact, the boolean satisfiability problem was the first problem proven to be NP complete in [21]. In his work, Cook showed how any nondeterministic Turing machine could be expressed as a series of logic clauses that are only satisfiable if and only if the corresponding Turing machine accepts its input. By doing so, he proved that the boolean satisfiability problem is equivalent to the class of problems, solvable by a nondeterministic Turing machine in polynomial time. SAT problems (and the variation Max-SAT) comprise a very well studied problem class with numerous highly effective heuristic solvers available.

#### Existing Hyperheuristic Research

Heuristic generation methods were applied to SAT in [38], where genetic programming was used in order to build new composite heuristics out of building blocks which were identified as parts from the well-known local search algorithms WalkSAT and Novelty. A very similar approach was used in [5], where a strongly-typed grammar-based genetic programming approach is used to evolve ”disposable”, problem instance specific heuristics.

#### Local Search Heuristics

The following local search heuristics for the Max-SAT problem were used for the development of this algorithm (as described in [54]):

- GSAT[96]
- HSAT[45]
- WalkSAT[95]
- Novelty[72]

**Mutation Heuristics**

The following mutation heuristics for the Max-SAT problem were used for the development of this algorithm (as described in [54]):

- flip a random variable
- flip a random variable from a broken clause
- flip a random variable from the set of variables with the highest gain
- flip the oldest non-flipped variable from the set of variables with the highest gain

## Bin-Packing

The one-dimensional bin-packing problem deals with packing objects of various weights into as few (fixed-capacity) bins as possible. Each object $j$ is assigned a weight $w_j$ and every bin is only allowed to be filled with objects as long as their total weight does not exceed the bin's capacity $c$. A mathematical formulation of the one-dimensional bin-packing problem taken from [71] can be seen in equation 3.2.

$$\begin{aligned}
\text{Minimise} \sum_{i=1}^{n} y_i & \\
\text{Subject to} \sum_{j=1}^{n} w_j x_{ij} \leq c y_i, & \qquad i \in N = 1, ..., n, \\
\sum_{i=1}^{n} x_{ij} = 1, & \qquad j \in N, \\
y_i \in \{0, 1\}, & \qquad i \in N, \\
x_{ij} \in \{0, 1\}, & \qquad i \in N, j \in N.
\end{aligned} \tag{3.2}$$

The binary variable $y_i$ indicates whether bin $i$ is non-empty, $x_{ij}$ is true iff object $j$ is contained in bin $i$ and $n$ is the total number of objects to be packed. A quadratic goal function (as described in [55]) is used, which favours bins that have very little remaining capacity.

### Existing Hyperheuristic Research

Heuristic generation methods have been applied to bin-packing in [14], where a genetic programming approach was able to autonomously evolve the well known first-fit heuristic often applied by humans. Additional research on the scalability of the evolved heuristics was conducted in [16]. A histogram-matching based genetic programming technique for evolving constructive heuristics is described in [87] and produces high quality solutions. In [67], the strongly related 0/1 knapsack problem with two separate objective functions is also addressed successfully with genetically evolved heuristics. A three dimensional version of this problem is investigated in [2], where evolved heuristics again produce results competitive to human level solutions. One of the earliest hyperheuristic research on the bin-packing problem was conducted in [94], where the classifier system XCS[108] is trained to select among various simple non-evolutionary heuristics.

### Local Search Heuristics

The following local search heuristics for the bin-packing problem were used for the development of this algorithm (as described in [55]):

- swap largest object from lowest filled bin with a smaller piece from a randomly selected bin
- take all objects from the lowest filled bin and repack them into the other bins

### Mutation Heuristics

The following mutation heuristics for the bin-packing problem were used for the development of this algorithm (as described in [55]):

- swap two random pieces
- split a highly filled bin into two bins
- empty the highest filled bins and repack them with a best-fit heuristic
- empty the lowest filled bins and repack them with a best-fit heuristic

## Permutation Flow-Shop

The permutation flow-shop problem is to find a sequence in which $n$ jobs can be scheduled on $m$ consecutive machines. Each job has to go through all of the available machines in the same order. A job can only be processed on one machine at once and also every machine can only work on a single job at a time. The ordering of the jobs has to be maintained throughout the whole process - that is, the order in which the

jobs enter the first machine is exactly the order in which they leave the last machine. The processing time of job $i$ on machine $j$ is denoted by $p_{ij}$ and the starting time with $start_{i,j}$. The goal is to find a permutation $\pi = \pi(1), ..., \pi(n)$, with $\pi(i)$ being the place in the job sequence assigned to job $i$, that minimises the completion time of the last job in the sequence. Equation 3.3 shows the calculation of the completion time $C_{\pi(i),j}$ of job $i$ on machine $j$.

$$\begin{aligned} & start_{\pi(i),j} = max\left\{start_{\pi(i),j-1}, start_{\pi(i-1),j}\right\}, \\ \text{with} & \\ & start_{\pi(0),j} = 0 \text{ and } start_{\pi(i),0} = 0, \\ & C_{\pi(i),j} = start_{\pi(i)} + p_{\pi(i),j}. \end{aligned} \tag{3.3}$$

**Existing Hyperheuristic Research**

In [105], a range of different genetic algorithms is used with a hybrid chromosome containing both, a schedule for the initial stage as well as a sequence of dispatching rules for the subsequent stages. This approach was termed "meta-hyper-heuristic" and gave encouraging results. An early hyperheuristic approach is used in [78], where a Heuristics Combination Method (HCM) is described which essentially performs heuristic selection. A variety of different approaches is examined, including some based on evolutionary algorithms. An interesting approach to the permutation flow-shop problem is described in [83], where a distributed agent-based system is used in which low-level heuristic agents cooperate by exchanging good solutions while synchronously searching the same solution space. Greedy and tabu-based strategies are used to decide about the specific low-level heuristic at each step.

**Local Search Heuristics**

The following local search heuristics for the permutation flow-shop problem were used for the development of this algorithm (as described in [104]):

- steepest descent local search
- first improvement local search
- random single local search step
- single first improvement local search step

**Mutation Heuristics**

The following mutation heuristics for the permutation flow-shop problem were used for the development of this algorithm (as described in [104]):

- randomly reinsert a job at a different position
- swap two randomly selected jobs
- shuffle the entire permutation
- create a new solution with the NEH algorithm[76]
- randomly shuffle a number of selected jobs
- remove a number of randomly selected jobs and reinsert them using NEH

### Personnel Scheduling

Personnel scheduling is an important task in a wide variety of businesses. It basically deals with deciding which employee should work on which day and in which shifts over a certain planning period. The details of this problem class are however very important for the definition of the problem and may be very different between application cases. Personnel scheduling is therefore only an umbrella term for a whole range of related problems, differing in their specific constraints and objectives. A constraint might for example be the total number of hours an employee is allowed to work per week. In another application this constraint might be an objective, where it is allowed for an employee to exceed a certain number of hours but this excess should be minimised. Each objective can also have a certain weight, which indicates how bad it is for a solution to not meet this objective. The test-environment at hand implements a framework which allows for a wide variety of personnel scheduling problems to be expressed. For details of the implementation see [30].

#### Existing Hyperheuristic Research

One of the early works that helped laying the foundations for the area of research on hyperheuristics was [25] and it applied three different categories of hyperheuristic approaches to a personnel scheduling problem class. They examined a random heuristic selection process, a greedy approach and one based on a choice function together with different acceptance criteria. Another variant of the personnel scheduling problem class is dealt with by the same author in [24], this time using a genetic algorithm for heuristic selection. Follow-up research was performed in [23], analysing a hyperheuristic approach dealing with a large number of available low-level heuristics. The work in [51] examines a guided genetic algorithm working on a chromosome representing the selection and the sequence of the underlying low-level heuristics. A simple heuristic is used to guide the injection or removal of gene groups from the chromosome. A personnel scheduling hyperheuristic employing a form of tabu-search among the low-level heuristics is introduced in [17]. The low-level heuristics are competing to be selected according to a rank-based metric inspired by reinforcement learning. A

stochastic search among low-level heuristics modelled after the simulated annealing technique is used in [6]. This pure heuristic selection process was found to give notable improvements over the best problem-specific metaheuristics known at that time.

#### Local Search Heuristics

The following local search heuristics for the personnel scheduling problem were used for the development of this algorithm (as described in [30]):

- first improvement hill-climber: swap shifts within a single employee's work schedule
- first improvement hill-climber: swap employees within a single shift assignment
- first improvement hill-climber: introduce or delete a group of shift assignments
- variable depth search [13]
- variable depth search with introducing and deleting moves

#### Mutation Heuristics

The following mutation heuristics for the personnel scheduling problem were used for the development of this algorithm (as described in [30]):

- randomly assign a number of new shifts
- randomly delete a number of shift-assignments keeping the solution feasible

### Travelling Salesman Problem

The travelling salesman problem is another very popular NP-hard combinatorial optimisation problem dating back to the 19th century, where the problem was defined by W. R. Hamilton and T. Kirkman. In its basic form, the goal is to find a shortest route among a number of cities, so that the travelling salesman visits each city exactly once and in the end returns to the starting point of his journey. The problem can conveniently be modelled using a weighted graph $G = (V, E)$, where the set of vertices $V = \{1, ..., n\}$ represents the cities and the edge set $E = \{(i, j) : i, j \in V, i < j\}$ corresponds to the possible paths between two cities, with each edge $(i, j)$ having an associated cost $c_{ij}$ which denotes the distance of the path. The solution to the travelling salesman problem is then to find a Hamiltonian cycle on G, which has the lowest total cost.

Due to the high practical importance of this problem class in applications ranging from transportation and logistics to printed circuit board manufacturing, there exists

a vast number of highly performant algorithms that use exact or heuristic methodologies. A very popular set of benchmark instances is available under the name of TSPLIB[89].

### Existing Hyperheuristic Research

The travelling salesman problem was approached with hyperheuristic techniques in [60], where a grammar-based genetic programming algorithm was used to construct heuristics that were able to rapidly solve benchmark instances from TSPLIB and produced results competitive with the best known results from the literature at that time. Another similar evolutionary approach known as Multi Expression Programming (MEP) was used for TSP in [80]. An MEP chromosome usually encodes several different computer programs, leading to a form of implicit parallelism. The results of this work mostly outperformed simple heuristics based on the nearest neighbour principle or on minimum spanning trees.

### Local Search Heuristics

The following local search heuristics for the travelling salesman problem were used for the development of this algorithm:

- two-opt local search with first improvement
- three-opt local search with first improvement
- two-opt local search with best improvement

### Mutation Heuristics

The following mutation heuristics for the travelling salesman problem were used for the development of this algorithm:

- greedily reinsert the cities of a deleted sub-sequence
- randomly reinsert a city at a different position within the tour
- swap two randomly selected cities
- randomly shuffle the tour
- randomly shuffle a sub-sequence of the tour
- flip the order of a sub-sequence of the tour

### Vehicle Routing Problem

The vehicle routing problem is a classic logistics problem of high practical importance. It was first introduced in 1959 by Dantzig[31] and shares some properties with the travelling salesman problem. The goal is to service a set of customers by visiting them with one of a set of available vehicles. A formal definition of the classical vehicle routing problem as found in [22] is the following: Let $G = (V, A)$ be a graph where $V = \{v_0, v_1, ..., v_n\}$ is a vertex set, and $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is an arc set. The special vertex $v_0$ represents a depot, while the remaining vertices correspond to customers. A cost matrix $c_{ij}$ and a travel time matrix $t_{ij}$ are associated with $A$. The vehicle routing problem is then defined on an undirected graph $G = (V, E)$, with the vertices $V$ and the edge set $E = \{(v_i, v_j) : v_i, v_j \in V, i < j\}$. Each customer has a non-negative demand $q_i$ and a service time $t_i$. A fleet of $m$ vehicles, each with the same capacity $Q$ is located at the depot $v_0$. The task is to construct a set of at most $m$ deliveries, such that the following conditions are fulfilled:

- each route starts and ends at $v_0$
- each customer is visited exactly once by exactly one vehicle
- the total demand of each route does not exceed $Q$
- the total duration of each route does not exceed a limit $D$
- the total routing cost is minimal

#### Existing Hyperheuristic Research

Five different vehicle routing problem variants are tackled with an adaptive large neighbourhood search (ALNS) in [86]. The adaptivity of this algorithm comes from a layer that performs heuristic selection among a number of insertion and removal heuristics that try to diversify and intensify the search. The results were very encouraging, as 183 new best solutions from a benchmark set containing 486 instances could be found. Another heuristic selection algorithm for the VRP was developed in [40]. There, the algorithm performs a hill-climbing search among pairs of constructive-perturbative heuristics. In [41], an evolutionary approach was applied to particularly hard VRP instances. Sequences of three different types of low-level heuristics (constructive, perturbative and noise heuristics) were evolved in order to create or improve upon partial solutions. The approach proved to result in very competitive solutions on a standard benchmark set.

#### Local Search Heuristics

The following local search heuristics for the vehicle routing problem were used for the development of this algorithm:

- two-opt local search
- local search using generalised insertion (GENI)[44]

**Mutation Heuristics**

The following mutation heuristics for the vehicle routing problem were used for the development of this algorithm:

- two-opt mutation
- or-opt[82] mutation
- delete a part of a tour within the area of a certain location and reinsert them using GENI
- delete a part of a tour within a certain timespan and reinsert them using GENI
- shift a part of a tour
- shift and mutate a part of a tour

## 3.3 Evaluation Method

Due to the somewhat novel area of cross-domain hyperheuristic algorithms, it is difficult to find existing research data for a fair comparison. Obviously for every problem instance there exists a multitude of specialised algorithms which are pretty much certain to achieve superior results. We therefore resorted to applying the same experimental settings as employed by the Cross-domain Heuristic Search Challenge 2011[1]. This challenge was specifically designed for algorithms that should perform well over a range of known and unknown independent problem domains and is therefore perfectly suited for judging the performance of the algorithm at hand.

The rules of the competition required the participants to make use of the HyFlex framework, which offers a concise interface for a clean hyperheuristic model. Solution candidates had to be written in pure Java, without the use of multi-threading or native code. A set of eight default hyperheuristics was made available prior to the contest, to be used as a baseline comparison during the development phase. These hyperheuristics were deemed to be of average quality by the organisers of the competition. In fact, after a few weeks of development time our new algorithm already outperformed even the best of the default hyperheuristics on almost all instances.

In order to provide a somewhat comparable metric for measuring the runtime of the participant's algorithms on their respective development systems, the organisers of the competition decided to provide a closed-source benchmark program. This

[1] http://www.asap.cs.nott.ac.uk/chesc2011

benchmark performed various typical operations within the HyFlex framework and measures their total execution time on the development system it is run on. The output of the program is a number of seconds, which should roughly be equivalent to 600 seconds on the final test platform for deciding the results of the competition. This benchmark was provided for both Windows and Linux operating systems, in 32 and 64 bit versions. Development and testing for the algorithm at hand was mainly performed on an Intel T9400 Core2 processor clocked at 2.53GHz, running a 32-bit Linux kernel. The benchmark program gave consistent ratings with no variance, although the same the benchmark run on the same machine under 32-bit Windows was a bit less stable. See table 3.1 for the output of the benchmarking program on 30 consecutive runs without any other system activity. Since the benchmarking program itself was only provided as a closed-source native binary, nothing can be said about its reliability but the stable result of the benchmarking program under Linux gave some confidence over the validity of the result. It turned out after the competition, that the benchmark seemingly overestimated the performance of the development machine significantly, leading to better results in the preliminary leaderboard.

| Platform | Min | Max | Avg | Stddev |
|---|---|---|---|---|
| 32-bit Linux | 715 | 715 | 715 | 0 |
| 32-bit Windows 7 | 692 | 745 | 702 | 15.6 |

**Table 3.1:** Benchmark results of 30 consecutive runs.

### Test Data

The test data for the evaluation of the algorithm was chosen among a number of different benchmark datasets described in [53] and in tables 3.2-3.7.

### Framework Description

The HyFlex framework (Hyperheuristics Flexible framework) as described in [12] and [10], offers an intuitive interface to utilise a set of given low-level search and mutation heuristics, easing the task of working purely on a high-level search strategy without any a priori knowledge about the problem instance or the heuristics available. It is an object-oriented framework implemented in the Java programming language, providing a concise way to access typical operations when working with hyperheuristics. It is built as a modular system, providing general-purpose methods for the interaction with separately implemented problem-specific heuristic modules. This separation follows the model depicted in figure 3.1 and allows the user to concentrate on the hyperheuristic part without having any knowledge about the underlying problem specific algorithms. The so-called domain barrier represents this abstraction, which is an important concept in hyperheuristic research. The only kind of information crossing the

| No. | Name | Source | Variables | Clauses |
|---|---|---|---|---|
| 0 | contest02-Mat26.sat05-457.reshuffled-07 | [27] | 744 | 2464 |
| 1 | hidden-k3-s0-r5-n700-01-S2069048075.sat05-488.reshuffled-07 | [27] | 700 | 3500 |
| 2 | hidden-k3-s0-r5-n700-02-S350203913.sat05-486.reshuffled-07 | [27] | 700 | 3500 |
| 3 | parity-games/instance-n3-i3-pp | [28] | 525 | 2276 |
| 4 | parity-games/instance-n3-i3-pp-ci-ce | [28] | 525 | 2336 |
| 5 | parity-games/instance-n3-i4-pp-ci-ce | [28] | 696 | 3122 |
| 6 | highgirth/3SAT/HG-3SAT-V250-C1000-1 | [3] | 250 | 1000 |
| 7 | highgirth/3SAT/HG-3SAT-V250-C1000-2 | [3] | 250 | 1000 |
| 8 | highgirth/3SAT/HG-3SAT-V300-C1200-2 | [3] | 300 | 1200 |
| 9 | MAXCUT/SPINGLASS/t7pm3-9999 | [3] | 343 | 2058 |
| 10 | jarvisalo/eq.atree.braun.8.unsat | [27] | 684 | 2300 |
| 11 | highgirth/3SAT/HG-3SAT-V300-C1200-4 | [3] | 300 | 1200 |

**Table 3.2:** Max-SAT instances

| No. | Name | Source | Capacity | Pieces |
|---|---|---|---|---|
| 0 | falkenauer/u1000-00 | [36] | 150 | 1000 |
| 1 | falkenauer/u1000-01 | [36] | 150 | 1000 |
| 2 | schoenfieldhard/BPP14 | [36] | 1000 | 160 |
| 3 | schoenfieldhard/BPP832 | [36] | 1000 | 160 |
| 4 | 10-30/instance1 | [56] | 150 | 2000 |
| 5 | 10-30/instance2 | [56] | 150 | 2000 |
| 6 | triples1002/instance1 | [56] | 1000 | 1002 |
| 7 | triples2004/instance1 | [56] | 1000 | 2004 |
| 8 | test/testdual4/binpack0 | [36] | 100 | 5000 |
| 9 | test/testdual7/binpack0 | [36] | 100 | 5000 |
| 10 | 50-90/instance1 | [56] | 150 | 2000 |
| 11 | test/testdual10/binpack0 | [36] | 100 | 5000 |

**Table 3.3:** Bin-packing instances

domain barrier are the fitness gains from the low-level heuristics as well as a coarse runtime estimation.

For every problem domain, the following problem specific components were provided:

- a set of suitable low-level heuristics, possibly with parameters
- an internal solution representation
- a scalar solution evaluation function
- methods to initialise, copy and compare solutions

| No. | Name | Source | Best Known | Staff | Shift Types | Days |
|---|---|---|---|---|---|---|
| 0 | BCV-3.46.1 | [29] | 3280 | 46 | 3 | 26 |
| 1 | BCV-A.12.2 | [29] | 1294 | 12 | 5 | 31 |
| 2 | ORTEC02 | [29] | 270 | 16 | 4 | 31 |
| 3 | Ikegami-3Shift-DATA1 | [57] | 2 | 25 | 3 | 30 |
| 4 | Ikegami-3Shift-DATA1.1 | [57] | 3 | 25 | 3 | 30 |
| 5 | Ikegami-3Shift-DATA1.2 | [57] | 3 | 25 | 3 | 30 |
| 6 | CHILD-A2 | [29] | 1111 | 41 | 5 | 42 |
| 7 | ERRVH-A | [29] | 2197 | 51 | 8 | 42 |
| 8 | ERRVH-B | [29] | 6859 | 51 | 8 | 42 |
| 9 | MER-A | [29] | 9915 | 54 | 12 | 42 |
| 10 | BCV-A.12.1 | [29] | 1294 | 12 | 5 | 31 |
| 11 | ORTEC01 | [29] | 270 | 16 | 4 | 31 |

**Table 3.4:** Personnel scheduling instances

| No. | Name | Source | Jobs | Machines |
|---|---|---|---|---|
| 0 | 100x20/1 | [99] | 100 | 20 |
| 1 | 100x20/2 | [99] | 100 | 20 |
| 2 | 100x20/3 | [99] | 100 | 20 |
| 3 | 100x20/4 | [99] | 100 | 20 |
| 4 | 100x20/5 | [99] | 100 | 20 |
| 5 | 200x10/2 | [99] | 200 | 10 |
| 6 | 200x10/3 | [99] | 200 | 10 |
| 7 | 500x20/1 | [99] | 500 | 20 |
| 8 | 500x20/2 | [99] | 500 | 20 |
| 9 | 500x20/4 | [99] | 500 | 20 |
| 10 | 200x20/1 | [99] | 200 | 20 |
| 11 | 500x20/3 | [99] | 500 | 20 |

**Table 3.5:** Pemutation flow-shop instances

The hyperheuristic is intentionally shielded from almost all low-level information about the available heuristics or the problem instance. The only information available to the hyperheuristic is:

- the number of available low-level heuristics
- a broad classification of each low-level heuristic
- optionally available parameters for the low-level heuristic
- the runtime information of previously applied heuristics
- a scalar solution quality measure

| No. | Name | Source | Cities |
|---|---|---|---|
| 0 | pr299 | [90] | 299 |
| 1 | pr439 | [90] | 439 |
| 2 | rat575 | [90] | 575 |
| 3 | u724 | [90] | 724 |
| 4 | rat783 | [90] | 783 |
| 5 | pcb1173 | [90] | 1173 |
| 6 | d1291 | [90] | 1291 |
| 7 | u2152 | [90] | 2152 |
| 8 | usa13509 | [90] | 13509 |
| 9 | d18512 | [90] | 18512 |
| 10 | 200x20/1 | [90] | 200 |
| 11 | 500x20/3 | [90] | 500 |

**Table 3.6:** Travelling salesman problem instances

| No. | Name | Source | Vehicles | Vehicle Capacity |
|---|---|---|---|---|
| 0 | Solomon/RC/RC207 | [97] | 25 | 1000 |
| 1 | Solomon/R/R101 | [97] | 25 | 200 |
| 2 | Solomon/RC/RC103 | [97] | 25 | 200 |
| 3 | Solomon/R/R201 | [97] | 25 | 1000 |
| 4 | Solomon/R/R106 | [97] | 25 | 200 |
| 5 | Homberger/C/C1-10-1 | [97] | 250 | 200 |
| 6 | Homberger/RC/RC2-10-1 | [97] | 250 | 1000 |
| 7 | Homberger/R/R1-10-1 | [97] | 250 | 200 |
| 8 | Homberger/C/C1-10-8 | [97] | 250 | 200 |
| 9 | Homberger/RC/RC1-10-5 | [97] | 250 | 200 |

**Table 3.7:** Vehicle routing problem instances

The classification of the low-level heuristic type can be one of: local search, mutation, crossover or ruin-recreate. This type information is made available only as a rough guideline - no guarantees about specific heuristic properties can be derived from this classification. It can however be expected that mutation heuristics usually result in a worse solution, whereas local search heuristics usually do not. Crossover type heuristics work with two solutions as input and produce one solution as output which usually shares some of the properties from the two input solutions. Ruin-recreate heuristics usually destroy a part of the solution and subsequently use a construction heuristic to rebuild a full solution.

The low-level heuristics were allowed to make use of none, one or both of the parameters named `intensityOfMutation` and `depthOfSearch`. Valid parameter values are floating point values between 0 and 1. As with the mentioned heuristic

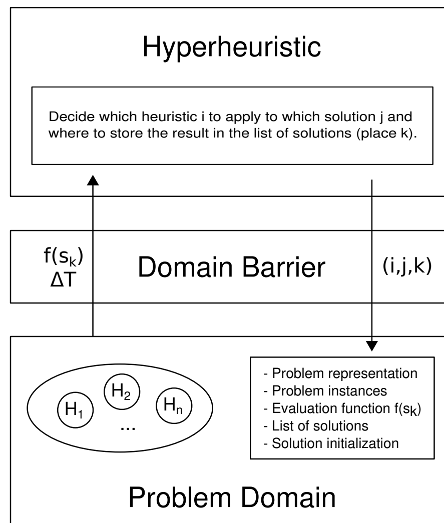


**Figure 3.1:** HyFlex abstraction model for hyperheuristics

classification, the names of these parameters are only a rough guideline and no guarantees are made about the actual usage of these parameters by the low-level heuristics. Usually, higher `intensityOfMutation` values lead to a greater part of the solution being changed, and higher `depthOfSearch` values lead to better but slower results from a local search heuristic.

Solutions are represented in a linear array of slots which has a predetermined size that can be set by the hyperheuristic. They are only ever addressed as indices within this array and cannot be accessed directly. After the solution array size has been set, individual solution slots can be initialised separately with a special call to the HyFlex framework. This call may use some kind of constructive heuristic to generate an initial solution. After the solutions have been initialised, repeated calls of the available low-level heuristic methods can be used to improve the fitness of the solution(s).

Apart from these main interaction methods with the problem specific part, HyFlex offers some utility functions to copy from one solution slot to another, to check two solution slots for equality, to explicitly get the fitness value of a solution slot and to gain a track record of past search performance as well as some coarse runtime estimation (measured in milliseconds).

### Scoring System

The scoring method of the competition was modelled after the Formula 1 point system in use until 2010: the top eight participating algorithms within each problem instance are awarded 10, 8, 6, 5, 4, 3, 2 and 1 points respectively. The rank of an algorithm is determined by the fitness value of the best solution found within a runtime limit of

600 seconds on a reference evaluation platform. If two or more algorithms produce the same fitness value to within a range of $10^{-6}$, the respective points are added together and split equally among the candidates. These scores are then summed over all 30 problem instances, thus resulting in the final ranking of the algorithms. Should two or more algorithms have the same final result score, priority is given to the algorithm which is more often ranked on the first place at the other instances. A separate program was provided which calculated the final score of an algorithm when competing against 8 default hyperheuristics which were deemed as of average quality.

As the score of an algorithm according to this system purely depends on the relative performance against a number of other competing algorithms, the organisers of the CHeSC 2011 provided a way for the participants to measure their relative performance in the run-up to the final competition. This leaderboard ranking was repeatedly published according to the raw fitness values on the test instances submitted by the participants on a voluntary basis. It has to be stressed of course, that the ranking of the leaderboard can only give a rough estimation of the algorithm's performance in the final competition, due to the following shortcomings:

- participation is purely voluntary (the real opponents in the actual challenge may be completely different)
- the results are obtained only from the test data provided initially and does not incorporate results on the hidden instances and the hidden domains
- the algorithms are run on different hardware and operating systems and only runtime limited by the result of the benchmark program

Especially the last factor had a rather heavy influence on the results of the leaderboard as was found out in the aftermath of the competition.

CHAPTER 4

# Algorithm Description

## 4.1 Overview

Following the classification of Burke et al. in [15], our algorithm is an online learning hyperheuristic, working mainly on heuristic selection. The novelty of the proposed approach lies in the repeated switching between two search variants, namely a serial search phase working only on a single solution and a systematic parallel search phase working with a set of different solutions at the same time. We further propose a grading mechanism for the ordering of the low-level heuristics and a selection strategy for the available mutation heuristics.

The following main components of the algorithm are executed repeatedly:

- Serial search
- Generation of mutated solutions
- Parallel search
- Selection of a new working solution

After an initialisation phase, in which the preliminary scores of the available heuristics are determined, the search continues by systematically executing the local-search heuristics in order of their respective quality scores. The splitting in a serial and a parallel search phase balances the focus of the search between exploration of new parts of the search space and the exploitation of the quality of the currently best working solution.

Throughout the search process, the performance and runtime characteristics of the low-level heuristics are measured and the scores responsible for their selection are updated accordingly. In addition to that, the overall search progress is monitored

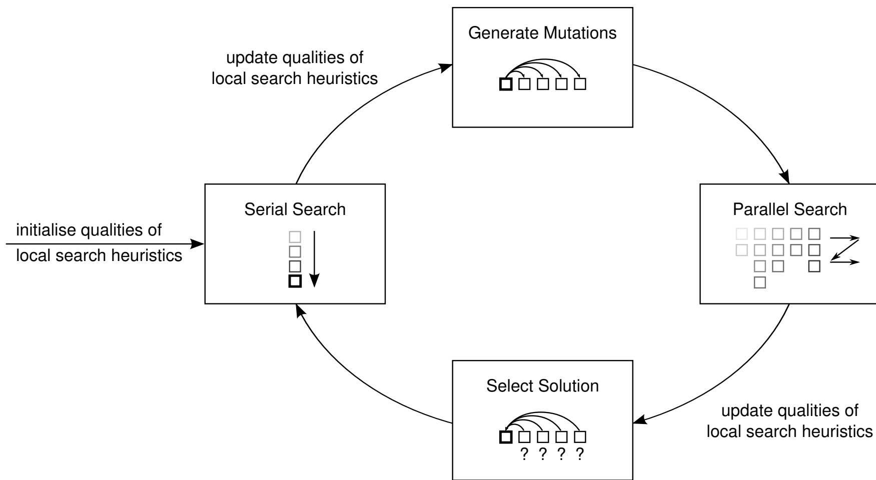


**Figure 4.1:** Execution cycle of the algorithm

continuously and mechanisms such as the temporary blocking of ineffective heuristics or the restart of the algorithm from the last best solution are applied. A tabu-list in form of a ringbuffer containing a determined number of already visited solutions is used to avoid cycles in the search process. The basic execution cycle of the different phases of the algorithm is illustrated in figure 4.1.

## 4.2 Detailed Description of the Algorithm

### Initialisation Phase

The algorithm begins with calculating preliminary quality-scores for all the local-search heuristics available for the given problem instance. It does this by initialising the first solution slot and applying the heuristics in turn, recording their gain and their required runtime. The heuristics are called with the highest possible parameter settings for the depth of search and the intensity of the mutation (where applicable). The quality-score $q_h$ for each heuristic $h$ is initially calculated as $q_h = g_h / t_h$ where $g_h$ denotes the gain as the difference between the solution value before and after the execution of $h$ and $t_h$ is the runtime of the local search heuristic. The best solution found during this initial phase is stored and returned as the working solution for the subsequent serial search phase.

### Serial Search Phase

The available local-search heuristics are applied sequentially to the current working solution, in order of decreasing qualities $q_h$. Whenever a solution is found that is either better or equally good but different, it is accepted as current working solution

and the search restarts with the best local-search heuristic. Additionally, a ringbuffer of limited size (determined experimentally) keeps track of the solutions visited so far. A solution which is already contained inside the ringbuffer is never accepted. The parameters for the depth of search and the intensity of mutation are set to random values before the application of each heuristic. The serial search phase ends whenever no further improvement could be found with all the available heuristics, therefore resulting in a locally optimal solution. See 4.3 for the corresponding pseudocode.

### Quality Updates

In predetermined time intervals, the qualities of the local-search heuristics are updated to reflect their performance during the whole search process. For each execution $i = \{1, ..., n\}$ of heuristic $h$, the runtime $t_{h,i}$ as well as the gain $g_{h,i}$ is logged. The gain denotes the difference between the function value of the solution before and after the application of $h$. The quality $q_h$ of each heuristic is then calculated using formula 4.1.

$$q_h = \frac{\sum_{i=1}^{n} H(g_{h,i})}{\sum_{i=1}^{n} t_{h,i}} \tag{4.1}$$

with

$$H(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

### Generation of Mutated Solutions

The available mutation and ruin-recreate heuristics are placed in a roulette-wheel reflecting their relative qualities. Initially all mutation heuristics have the same chance of being selected (i.e. the same quality). After each execution $i$ of the mutation heuristic $h$, its quality $q'_h$ is set to $q'_h = \sum_{i=1}^{n} g'_{h,i}$ where $g'_{h,i}$ denotes the gain of the mutation heuristic $h$ as the difference between the function value of the solution before the application of $h$ and after the application of $h$ and a subsequent optimisation run with the local search heuristics. This optimisation run is effectively part of the parallel search phase and ensures that the solution is in a local optimum with respect to all local search heuristics. This special quality measure for mutation heuristics is used, because the immediate change in the function value after its application does not allow to determine the usefulness of a given mutation heuristic.

With the current working solution as input, a set of mutated offsprings is generated by applying a selection of mutation heuristics chosen by a repeated run of the roulette-wheel process. The mutation intensity is set using the formula $r \cdot (1 - t_{used}/t_{total})$, where $r$ is a random number between 0 and 1, $t_{used}$ is the number of milliseconds since the start of the algorithm and $t_{total}$ is the total allotted runtime in milliseconds. This reduces the probability for large mutations towards the end of the search. A pseudocode implementation is given in 4.4.

### Parallel Search Phase

Starting from the set of mutated solutions, the parallel search phase begins to work on the candidate solutions one after another. It begins with applying the best local-search heuristic to the first candidate. If an improvement is found, the new solution is accepted, otherwise it is discarded. Afterwards the search continues with the next solution slot and the local-search heuristic scheduled for this slot. Whenever a global improvement is found (i.e. the result is better than the currently best found solution so far), it is immediately accepted as the working solution for the next serial search phase and the parallel search is aborted. Otherwise the search goes on until all solutions have reached a local optimum with respect to all available local-search heuristics. See algorithm 4.7 for a pseudocode implementation.

### Working Solution Selection

Assuming no global improvement was found during the parallel search phase, a solution is selected from the pool of solutions containing the locally optimal output from the serial search phase as well as the parallel search phase. Solutions with a better quality have higher probability of being selected. If all solutions in the set are contained in the ringbuffer, a random mutation will be applied to the best solution until the result is not contained in the buffer anymore.

### Additional Mechanisms

A number of additional mechanisms were implemented in order to improve the performance of the algorithm and make sure the search does not get stuck.

#### Skipping Inefficient Heuristics

If a local-search heuristic is found to be ineffective it is skipped at both the serial and the parallel search phase with a 50% chance. The definition of an ineffective heuristic is, that the rate of successful applications is below 1%. Note, that this also helps to weed out local search heuristics that are ”masked” by another (better) local search heuristic, i.e. they never improve a solution once the other (better) heuristic has failed to improve the same solution. This effect is the result of the way the local search heuristics are applied: a heuristic of lower quality never gets the chance to work on a solution that a higher quality heuristic has managed to improve.

#### Restarting the Search

Whenever the search continues for a certain amount of time or a certain number of search iterations, producing only solutions which are worse than a given threshold (relative to the currently best solution), the search continues with the generation of

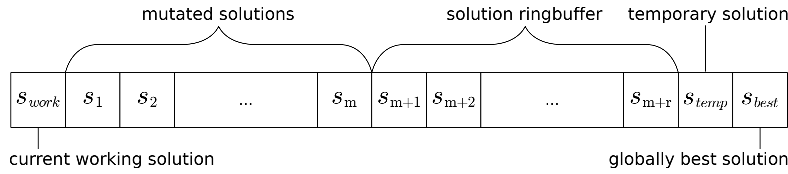


**Figure 4.2:** Internal structure of solution array, with $m$ mutated solutions and a ringbuffer of size $r$

mutated solutions from the currently best solution. This helps to avoid a diverging search progress, where the solution quality only gets worse.

#### Hashed Solution Ringbuffer

Because the HyFlex framework does not allow for direct solution access, it is not straight-forward to implement an efficient solution ringbuffer. The naive approach to perform a linear search within the ringbuffer in order to determine if a solution is already contained can be too time-consuming. To remedy this situation, a chaining hashtable using the function value of a solution as key for the hash function is implemented to speed up the access to the ringbuffer.

#### Constant Assignments

The proposed algorithm uses a number of constants which were determined experimentally. The following assignments were found to work well for the given problem classes and test environment:

- number of mutated solutions: 7
- size of the ringbuffer: 50
- quality update interval: 5000 milliseconds
- threshold for search restart: 150% of the currently best solution
- time interval outside the threshold to trigger search restart: 6000 milliseconds
- number of search iterations outside the threshold to trigger search restart: 6

## 4.3 Pseudocode Implementation

In this Section we give an approximate pseudocode version of our proposed hyperheuristic HAHA. Note, that only the main functionality can be seen from this implementation and that especially the additional mechanisms described above have been omitted for clarity.

Algorithm 4.1 shows the main problem solving method of the hyperheuristic which maintains the basic execution cycle shown in 4.1. The working of the serial and parallel search phase can be seen in 4.3 and 4.7 respectively. Algorithm 4.4 outlines the generation of the mutated solutions as input for the parallel search and 4.6 shows the selection of the working solution for the subsequent iteration of the execution cycle. The remaining methods used in this pseudocode version are the initialisation of the heuristic qualities 4.2, as well as the periodic update of these qualities 4.5.

**Algorithm 4.1:** HAHA main solving method

1: HAHA
2: $LS \leftarrow$ available local search heuristics sorted by decreasing quality $q$
3: $b \leftarrow$ create ringbuffer
4: $s \leftarrow$ create solution array
5: $s_{work} \leftarrow$ INITIALISELSQUALITIES()
6: $s_{best} \leftarrow s_{work}$
7: **while** time not expired **do**
8:   b.add($s_{work}$)
9:   SERIALSEARCH($s_{work}$)
10:   **for** $i = 1 \rightarrow 7$ **do**
11:     $s_i \leftarrow$ GENERATEMUTATION($s_{work}$)
12:   **end for**
13:   PARALLELSEARCH($s_{1..7}$)
14: **end while**
15: **return** $s_{best}$

**Algorithm 4.2:** Initialisation of quality measures for the local search heuristics

1: INITIALISELSQUALITIES
2: $s_{work} \leftarrow$ generate random initial solution
3: **for** $i = 1 \rightarrow |LS|$ **do**
4:   $t_i \leftarrow t_{now}$
5:   $s_i \leftarrow$ applyHeuristic($LS_i, s_{work}$)
6:   $t_i \leftarrow t_{now} - t_i$
7:   $q_i \leftarrow t_i / (s_{work} - s_i)$
8:   **if** getFitness($s_i$) $<$ getFitness($s_{best}$) **then**
9:     $s_{best} \leftarrow s_i$
10:   **end if**
11: **end for**
12: **return** $s_{best}$

**Algorithm 4.3:** Serial search using a single solution $s_{work}$

1: SERIALSEARCH($s_{work}$)
2: **for** $i = 1 \rightarrow |LS|$ **do**
3: setDepthOfSearch(random(0..1))
4: $s_{temp} \leftarrow$ applyHeuristic($LS_i$, $s_{work}$)
5: **if** time to update the qualities **then**
6: UPDATELSQUALITIES()
7: **end if**
8: **if** NOT b.contains($s_{temp}$) AND (getFitness($s_{temp}$) $<$ getFitness($s_{work}$) OR (getFitness($s_{temp}$) $=$ getFitness($s_{work}$) AND $s_{work} \neq s_{temp}$)) **then**
9: $s_{work} \leftarrow s_{temp}$
10: b.add($s_{work}$)
11: $i \leftarrow 1$
12: **end if**
13: **end for**

**Algorithm 4.4:** Generate a mutated solution based on $s_{work}$

1: GENERATEMUTATIONS($s_{work}$)
2: setIntensityOfMutation(random(0..$(1 - t_{used}/t_{total})$))
3: $h_m \leftarrow$ select mutation heuristic with roulette wheel of mutation qualities
4: $s_m \leftarrow$ applyHeuristic($h_m, s_{work}$)
5: **return** $s_m$

**Algorithm 4.5:** Update the quality measures for the local search heuristics

1: UPDATELSQUALITIES
2: **for** $i = 1 \rightarrow |LS|$ **do**
3: $imp_i \leftarrow$ number of times heuristic $LS_i$ has improved the solution
4: $runtime_i \leftarrow$ total runtime of heuristic $LS_i$
5: $q_i = imp_i / runtime_i$
6: **end for**

**Algorithm 4.6:** Select one of the best solutions from the given solution set $s_{1..7}$

1: SELECTSOLUTION($s_{1..7}$)
2: sort solution $s_{1..7}$ in decreasing order of their fitnesses
3: **for** $i = 1 \rightarrow 7$ **do**
4: **if** random(0..1)$<$ 0.8 **then**
5: **return** $s_i$
6: **end if**
7: **end for**
8: **return** $s_7$

**Algorithm 4.7:** Parallel search operating on a set of solutions $s_{1..7}$ simultaneously

1: PARALLELSEARCH
2: **for** $i = 1 \rightarrow 7$ **do**
3:   // start with all working heuristics set to the best LS heuristic (index 1)
4:   $wh_i \leftarrow 1$
5: **end for**
6: **repeat**
7:   $candidatesLeft \leftarrow false$
8:   **for** $i = 1 \rightarrow 7$ **do**
9:     **if** $wh_i \leq |LS|$ **then**
10:       setDepthOfSearch(random(0..1))
11:       $s_{temp} \leftarrow$ applyHeuristic($LS_{wh_i}$, $s_i$)
12:       **if** getFitness($s_{temp}$) $<$ getFitness($s_{best}$) **then**
13:         $s_{best} \leftarrow s_{temp}$
14:         $s_{work} \leftarrow s_{temp}$
15:         b.add($s_{temp}$)
16:         **return**
17:       **else**
18:         **if** getFitness($s_{temp}$) $<$ getFitness($s_i$) **then**
19:           $s_i \leftarrow s_{temp}$
20:           b.add($s_{temp}$)
21:           // continue with best LS heuristic (index 1)
22:           $wh_i \leftarrow 1$
23:         **else**
24:           // continue with the next best LS heuristic
25:           $wh_i \leftarrow wh_i + 1$
26:           $candidatesLeft \leftarrow true$
27:         **end if**
28:       **end if**
29:     **end if**
30:   **end for**
31: **until** not $candidatesLeft$
32: **if** b.contains($s_{1..7}$) **then**
33:   randomly mutate $s_{work}$
34: **else**
35:   $s_{work} \leftarrow$ SELECTSOLUTION($s_{1..7}$)
36: **end if**

CHAPTER 5

# Experimental Results

In this Chapter we show experimental results for the problems defined in Section 3.2. These are the official scores published by the organisers of the competition[1]. The evaluation instances were randomly chosen for the contest and include two hidden instances from each domain which were not available prior to the submission date for the competition. The values in the result tables represent the median solution value per instance of 31 subsequent runs with different random seeds. The runtime was limited to 600 seconds CPU-time and the evaluation of each algorithm was performed on the same machine in order to make the results comparable.

Tables 5.1 through 5.6 show the results for each of the problem domain separately and table 5.7 presents the overall ranking of the final competition for each of the 20 competitors. In figures 5.1 through 5.6 you see graphs of the 15 best algorithms for each domain and the total scores in 5.7. Note that the scoring system assigns a maximum of 50 points for each of the domains (5 instances with no more than 10 points each).

In figure 5.1 we see that the top 3 algorithms are very close together with our algorithm ranked on third place. This is probably due to the low integer solution values of the problem instances, leaving little room for significantly better results and producing many algorithms with identical fitness values, which in turn leads to a somewhat distorted view because of the ranking performed by the scoring system.

In the bin-packing domain we observe the really outstanding performance of the AdapHH algorithm, which dominates the field (see figure 5.2). The fact that this is the only domain where continuous improvements are easily accessible through the low-level heuristics may be a reason for the big advantage of one algorithm over all the others. While this domain shows the great efficiency of AdapHH, it also reveals a possible weakness in the quality grading scheme of our algorithm because there is

[1] `http://www.asap.cs.nott.ac.uk/chesc2011/results.html`

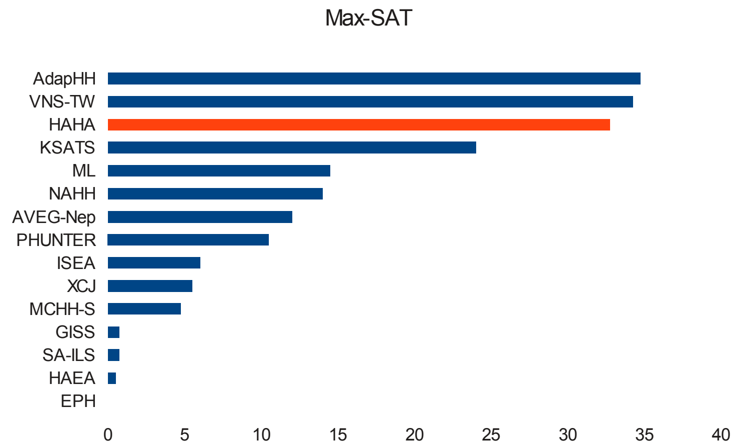


**Figure 5.1:** Max-SAT results

no quality distinction among equally fast, always improving low-level heuristics. Our quality measure which proved to be rather robust with the other problem domains leads to difficulties in the bin-packing class.

Figure 5.3 shows a rather evenly distributed picture of the scores, which might be due to the fact that this is the slowest problem class where individual low-level heuristics may take many seconds to complete. Therefore the results of this domain are probably the most random because every hyperheuristic only has a very small number of attempts to apply the low-level heuristics. This observation is further supported by the fact that the otherwise superior AdapHH only achieves a rather poor 10th rank. Despite the difficulties of this domain, our algorithm manages to perform very well within this problem class.

Permutation flow-shop results in 5.4 show that almost half of the algorithms do not manage to score any points in this domain, giving rise to the suspicion that its fitness landscape is very hard for a large part of the different hyperheuristic approaches. A similar picture can be seen with the results of the travelling salesman problem in 5.5.

Interestingly, the results of the vehicle routing problem in 5.6 are very different from the travelling salesman instances, although the two domains should in theory exhibit similar characteristics. This is a good example where different low-level heuristics can change the problem environment completely even if the problem themselves are very much alike. This is also reflected in the significant differences of the rankings from the algorithms between those two domains.

| Algorithm | Inst 3 | Inst 5 | Inst 4 | Inst 10 | Inst 11 |
|---|---|---|---|---|---|
| VNS-TW | **3.0** | **3.0** | **2.0** | **3.0** | 10.0 |
| SA-ILS | 13.0 | 23.0 | 12.0 | 15.0 | 9.0 |
| DynILS | 23.0 | 56.0 | 37.0 | 31.0 | 19.0 |
| ML | 5.0 | 10.0 | 3.0 | 9.0 | 8.0 |
| AdapHH | **3.0** | 5.0 | **2.0** | 3.0 | 8.0 |
| KSATS | 4.0 | 7.0 | **2.0** | 4.0 | 9.0 |
| EPH | 7.0 | 11.0 | 6.0 | 14.0 | 13.0 |
| GenHive | 16.0 | 44.0 | 31.0 | 19.0 | 14.0 |
| PHUNTER | 5.0 | 11.0 | 4.0 | 9.0 | 8.0 |
| ACO-HH | 11.0 | 35.0 | 9.0 | 17.0 | 13.0 |
| **HAHA** | **3.0** | 4.0 | **2.0** | 5.0 | 8.0 |
| ISEA | 5.0 | 11.0 | 4.0 | 9.0 | 11.0 |
| GISS | 16.0 | 21.0 | 13.0 | 17.0 | 9.0 |
| SelfS | 13.0 | 36.0 | 14.0 | 14.0 | 10.0 |
| XCJ | 6.0 | 8.0 | 5.0 | 9.0 | 10.0 |
| AVEGNep | 8.0 | 10.0 | 5.0 | 9.0 | **7.0** |
| MCHH-S | 8.0 | 14.0 | 8.0 | 8.0 | 9.0 |
| HAEA | 6.0 | 12.0 | 5.0 | 12.0 | 11.0 |
| Ant-Q | 23.0 | 52.0 | 38.0 | 27.0 | 14.0 |
| NAHH | 8.0 | 10.0 | 4.0 | 9.0 | **7.0** |

**Table 5.1:** Max-SAT results

## 5.1 Discussion of Results

The results show that our algorithm performs very well with the Max-SAT and the personnel scheduling problem classes. This is very interesting, because these two domains are conceptionally very different. While the heuristics for Max-SAT are very fast and complete within the millisecond range, the heuristics of personnel scheduling take a long time to complete, often requiring over 10 seconds for a single call. Because of this result it is clear that a ranking of local search heuristics can be very effective both with very fast and very slow heuristics. The worst results for the algorithm are in the bin-packing and travelling salesman domains. The bin-packing problems have very fast heuristics as well but also heuristics that seem to be able to result in very small improvements throughout the whole search process. This might be a case where the quality measure used for the ranking of the local search heuristics is suboptimal. The bin-packing domain was the only case where this phenomenon was observed.

The travelling salesman problem is strongly related to the vehicle routing problem, yet the ranking for our algorithm are rather far apart. Both domains share the theoretical concept of finding short cycles in graphs, both use moderately fast heuristics and the distribution of the results has similar properties as well. One thing that might have

| Algorithm | Inst 7 | Inst 1 | Inst 9 | Inst 10 | Inst 11 |
|---|---|---|---|---|---|
| VNS-TW | 0.03696 | 0.00715 | 0.01671 | 0.10878 | 0.02776 |
| SA-ILS | 0.07873 | 0.01153 | 0.01458 | 0.11039 | 0.02958 |
| DynILS | 0.04027 | 0.00767 | 0.01016 | 0.10872 | 0.01285 |
| ML | 0.04214 | 0.00753 | 0.01456 | 0.10852 | 0.02182 |
| AdapHH | **0.01607** | 0.00360 | **0.00356** | **0.10828** | **0.00354** |
| KSATS | 0.01923 | 0.00780 | 0.01149 | 0.10892 | 0.02199 |
| EPH | 0.05042 | 0.00360 | 0.01127 | 0.10866 | 0.02238 |
| GenHive | 0.02994 | 0.00708 | 0.01037 | 0.10859 | 0.02286 |
| PHUNTER | 0.04787 | 0.00360 | 0.02012 | 0.10908 | 0.03948 |
| ACO-HH | 0.04771 | 0.00320 | 0.00388 | 0.10986 | 0.01486 |
| **HAHA** | 0.08829 | 0.00726 | 0.01450 | 0.11023 | 0.02790 |
| ISEA | 0.03422 | **0.00328** | 0.00365 | 0.10862 | 0.00640 |
| GISS | 0.06917 | 0.00837 | 0.03218 | 0.11259 | 0.05922 |
| SelfS | 0.06642 | 0.00736 | 0.01441 | 0.10968 | 0.02391 |
| XCJ | 0.02201 | 0.01145 | 0.01569 | 0.10856 | 0.02850 |
| AVEGNep | 0.08737 | 0.00773 | 0.01807 | 0.11139 | 0.03750 |
| MCHH-S | 0.06225 | 0.00729 | 0.01459 | 0.10976 | 0.02861 |
| HAEA | 0.04522 | 0.00363 | 0.01379 | 0.10873 | 0.02400 |
| Ant-Q | 0.04909 | 0.01650 | 0.02102 | 0.10990 | 0.03765 |
| NAHH | 0.05504 | 0.00347 | 0.00473 | 0.10878 | 0.00554 |

**Table 5.2:** Bin-packing results

contributed to these seemingly contradictory results is the fact that the solutions for these instances are very close together, with the average results from the contestants always being within a few percent of the best result. Because of this, the ranking of the algorithms together with the integer scoring system might be not ideal for this domains, leading to a slightly skewed picture.

Overall, the results are competitive and certainly justify further research on this approach. Especially the fact that it managed to perform very good on two totally different problem domains is an encouraging result with respect to the original goal of cross-domain applicability. The competition also helped to point out some weaknesses of the quality measure which will be addressed in future versions of the algorithm. Another result from the competition is that the specific details of the employed scoring system can have a high influence on the final results and that it is not equally well suited for the different problem domains. A possible improvement for the comparisons of the result would be to switch from an ordinal scale to a ratio scale that expresses the results as normalised real values in the range of 0..1. A suitable exponential weighting function could then be applied to give the best algorithms an additional advantage (as implemented in the F1-scoring system), while still maintaining the requirement of having a fixed sum of scores for each of the problem instances.

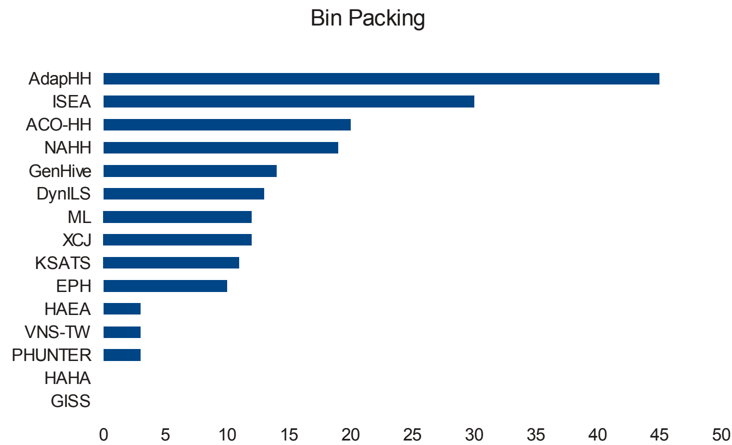


**Figure 5.2:** Bin-packing results

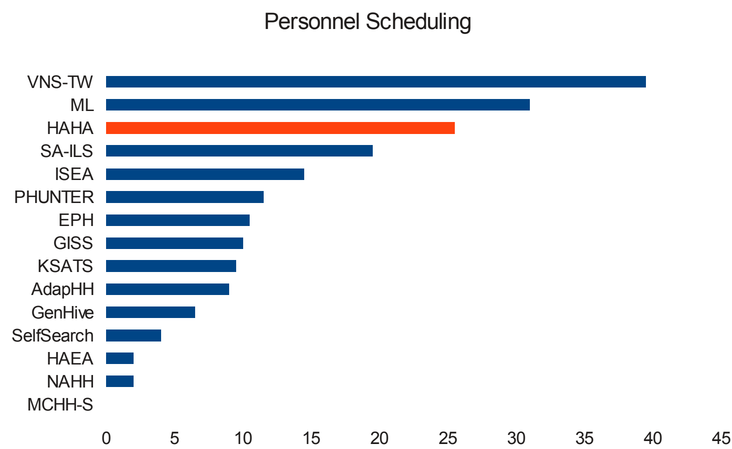


**Figure 5.3:** Personnel scheduling results

Such a continuous rating system would have far superior statistical properties over the currently used scheme.

## 5.2 Description of Competing Algorithms

The following Section describes the approach by the four top-scoring algorithms in the competition. A short overview about the inner workings as well as a short performance comparison is given.

| Algorithm | Inst 5 | Inst 9 | Inst 8 | Inst 10 | Inst 11 |
|---|---|---|---|---|---|
| VNS-TW | 19 | 9628 | **3223** | 1590 | 320 |
| SA-ILS | 20 | 9750 | 3228 | 1625 | 340 |
| DynILS | 33 | 9893 | 3324 | 1870 | 465 |
| ML | **18** | 9812 | 3228 | 1605 | **315** |
| AdapHH | 24 | 9667 | 3289 | 1765 | 325 |
| KSATS | 22 | 9681 | 3241 | 1640 | 355 |
| EPH | 22 | 10074 | 3232 | 1615 | 345 |
| GenHive | 21 | 12708 | 3274 | 1727 | 330 |
| PHUNTER | 25 | 10136 | 3255 | 1595 | 320 |
| ACO-HH | 26 | 11212 | 3346 | 1760 | 355 |
| **HAHA** | 21 | 9666 | 3236 | **1558** | 335 |
| ISEA | 20 | 9966 | 3308 | 1660 | **315** |
| GISS | 25 | **9625** | 3294 | 1785 | 370 |
| SelfS | 26 | 9803 | 3249 | 1635 | 350 |
| XCJ | 30 | 33390 | 3277 | 1658 | 380 |
| AVEGNep | 26 | 10230 | 3283 | 1765 | 360 |
| MCHH-S | 32 | 13297 | 3344 | 1785 | 370 |
| HAEA | 25 | 9795 | 3266 | 1699 | 345 |
| Ant-Q | 33 | 73535 | 3348 | 1970 | 425 |
| NAHH | 27 | 9827 | 3246 | 1644 | 345 |

**Table 5.3:** Personnel scheduling results

### AdapHH

The adaptive hyper-heuristic AdapHH as described in [74] produced by far the best results in the final competition. It uses an adaptive dynamic heuristic set (ADHS) that keeps track of the performance of the individual heuristics and excludes all but the best performing heuristics after a certain number of iterations. The metric to determine the quality of a heuristic is shown in formula 5.1.

$$\begin{aligned} p_i = & w_1 \left[ (C_{p,best}(i) + 1)^2 (t_{remain} / t_{p,spent}(i)) \right] \times b + \\ & w_2 (f_{p,imp}(i) / t_{p,spent}(i)) - w_3 (f_{p,wrs}(i) / t_{p,spent}(i)) + \\ & w_4 (f_{imp}(i) / t_{spent}(i)) - w_5 (f_{wrs}(i) / t_{spent}(i)) \end{aligned}$$

$$b = \begin{cases} 1, & \sum_{i=0}^{n} C_{p,best}(i) > 0 \\ 0, & otherwise \end{cases} \tag{5.1}$$

In this formula, $C_{p,best}(i)$ is the number of new best solutions found, $f_{imp}(i)$ and $f_{wrs}(i)$ denote the sum of improvements and worsenings. $f_{p,imp}(i)$ and $f_{p,imp}(i)$ and

| Algorithm | Inst 1 | Inst 8 | Inst 3 | Inst 10 | Inst 11 |
|---|---|---|---|---|---|
| VNS-TW | 6251 | 26803 | 6328 | 11376 | **26602** |
| SA-ILS | 6336 | 26886 | 6390 | 11514 | 26703 |
| DynILS | 6269 | 26875 | 6365 | 11419 | 26670 |
| ML | 6245 | **26800** | **6323** | 11384 | 26610 |
| AdapHH | **6240** | 26814 | 6326 | **11359** | 26643 |
| KSATS | 6292 | 26860 | 6366 | 11466 | 26683 |
| EPH | 6250 | 26816 | 6347 | 11397 | 26640 |
| GenHive | 6279 | 26835 | 6366 | 11434 | 26648 |
| PHUNTER | 6253 | 26858 | 6350 | 11388 | 26677 |
| ACO-HH | 6249 | 26904 | 6353 | 11393 | 26724 |
| **HAHA** | 6269 | 26850 | 6353 | 11419 | 26663 |
| ISEA | 6262 | 26844 | 6366 | 11419 | 26663 |
| GISS | 6329 | 26979 | 6385 | 11516 | 26758 |
| SelfS | 6287 | 26859 | 6369 | 11443 | 26678 |
| XCJ | 6271 | 26910 | 6366 | 11481 | 26710 |
| AVEGNep | 6322 | 26952 | 6379 | 11507 | 26743 |
| MCHH-S | 6336 | 26937 | 6397 | 11527 | 26716 |
| HAEA | 6261 | 26826 | 6353 | 11408 | 26651 |
| Ant-Q | 6358 | 26971 | 6407 | 11545 | 26792 |
| NAHH | 6245 | 26885 | **6323** | 11383 | 26671 |

**Table 5.4:** Permutation flow-shop results

$f_{p,wrs}(i)$ refer to the same values, but only aggregated over a single search iteration (phase). $t_{remain}$ denotes the remaining time of the algorithm, $t_{spent}(i)$ and $t_{p,spent}(i)$ mark the time spent by heuristic $i$ from the start of the algorithm or the start of the current phase respectively. The weights $w_{1,2,3,4,5}$ are set in a decreasing manner and are sufficiently far apart so that the influence of the individual terms is ranked in order of their importance. The performance measure $p_i$ is then used to rank the heuristics and those from the worse half of the ranking are excluded for a number of search phases (tabu duration). Each time a heuristic is to be applied, one is selected from the adaptive set with probability $pr_i$ shown in formula 5.2. At the end of each search phase some additional heuristics are excluded based on the distribution of a second quality metric for each heuristic.

$$\begin{aligned} pr_i &= ((C_{best}(i) + 1)/t_{spent})^{(1+3tf^3)} \\ tf &= (t_{exec} - t_{elapsed})/t_{exec} \end{aligned} \tag{5.2}$$

AdapHH also employs a form of relay hybridisation, that tries to determine pairs of heuristics that were found to be effective when applied consecutively. The parameters for the mutation intensity and the search depth are continuously updated according to

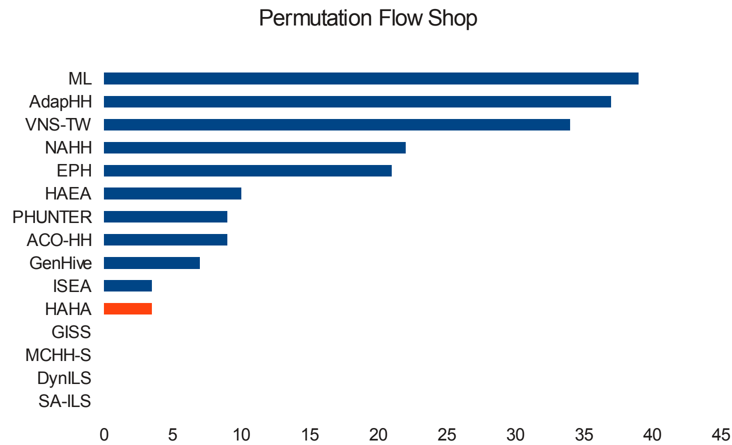


**Figure 5.4:** Permutation flow-shop results

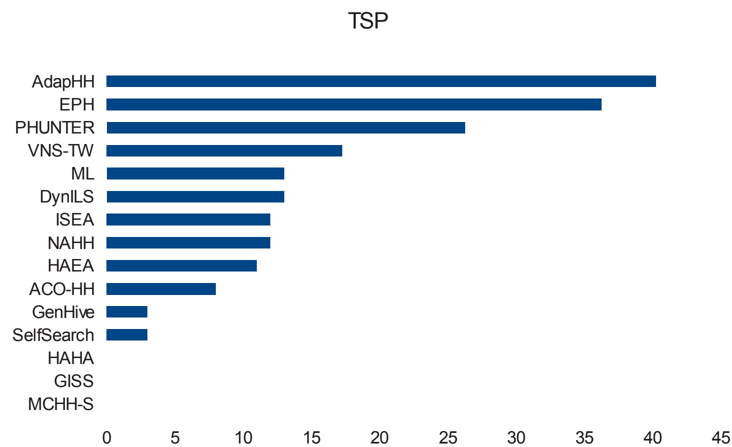


**Figure 5.5:** Travelling salesman problem results

a reward-penalty strategy for each heuristic. A sophisticated move acceptance criterion called "Adaptive Iteration Limited List-based threshold Accepting" (AILLA) is used that takes the fitness values of previously found new best solutions into account in order to adaptively set the acceptance threshold.

Finally, a completely new solution is initialised and used to continue the search, whenever a certain threshold level is reached. This threshold level is set according to the remaining execution time, the cost of reinitialisation (runtime of the initialisation heuristic) and the possibility of finding a new best solution after reinitialising.

AdapHH gave outstanding results in the final competition - it is ranked first in the

| Algorithm | Inst 0 | Inst 8 | Inst 2 | Inst 7 | Inst 6 |
|---|---|---|---|---|---|
| VNS-TW | **48194** | 21042675 | 6819 | 67378 | **54028** |
| SA-ILS | 49046 | 21281226 | 6994 | 70614 | 57607 |
| DynILS | **48194** | 20987358 | 6823 | 67308 | 54100 |
| ML | **48194** | 21093828 | 6820 | 66893 | 54368 |
| AdapHH | **48194** | **20822145** | **6810** | 66879 | 53099 |
| KSATS | 48578 | 21557455 | 6947 | 72027 | 58738 |
| EPH | **48194** | 21064606 | 6811 | **66756** | 52925 |
| GenHive | 48271 | 21083157 | 6868 | 67236 | 56022 |
| PHUNTER | **48194** | 21246427 | 6813 | 67136 | 52934 |
| ACO-HH | 48200 | 21137472 | 6851 | 67202 | 53428 |
| **HAHA** | 48414 | 21291914 | 6917 | 69324 | 56039 |
| ISEA | **48194** | 20868203 | 6832 | 67282 | 54129 |
| GISS | 49010 | 21651052 | 7001 | 72630 | 59804 |
| SelfS | 49043 | 21040810 | 6984 | 69646 | 56647 |
| XCJ | 48412 | 21162559 | 6884 | 68005 | 54967 |
| AVEGNep | 48639 | 21520601 | 6969 | 70194 | 57998 |
| MCHH-S | 49412 | 21504030 | 6997 | 70685 | 57836 |
| HAEA | **48194** | 20925949 | 6824 | 67488 | 54144 |
| Ant-Q | 49613 | 21277953 | 7016 | 69987 | 55314 |
| NAHH | **48194** | 20971771 | 6841 | 67418 | 53097 |

**Table 5.5:** Travelling salesman problem results

Max-SAT, the bin-packing and the TSP instances as well as close second in the flow-shop category. The VRP and especially the personnel scheduling domains however proved to be a harder challenge for this algorithm. As personnel scheduling is a very slow domain, where the low-level search heuristics require a long time to complete, this could be an indication that the mechanisms of AdapHH require more iterations to be effective than our algorithm.

### VNS-TW

The hyperheuristic VNS-TW by Hsiao et. al. uses an iterated local search strategy with a three level ranking system of the local search heuristics. The algorithm starts with first perturbing an inital solution with a randomly selected mutation or ruin-recreate heuristic before it is improved with local search. After each application of a local search heuristic, their ranks are updated to be one of the three available levels: $-1, 0, 1$. As soon as an improvement to the current solution is found, the ranks of all heuristics are set to 1. If the local search heuristic did not change the fitness of the solution its rank is set to 0 and to $-1$ if it also did not change the solution at all. The local search phase is aborted as soon as all heuristics have the rank $-1$ or a number of consecutive steps

| Algorithm | Inst 6 | Inst 2 | Inst 5 | Inst 1 | Inst 9 |
|---|---|---|---|---|---|
| VNS-TW | 76147 | 13367 | 148206 | 21642 | 149132 |
| SA-ILS | 64185 | 13390 | 162642 | 20667 | 152271 |
| DynILS | 69798 | 14359 | 149869 | 21654 | 150060 |
| ML | 80671 | 13329 | **145333** | 20654 | 148975 |
| AdapHH | 60900 | 13347 | 148516 | 20656 | 148689 |
| KSATS | 64495 | 13296 | 156577 | 20655 | **147124** |
| EPH | 74715 | 13335 | 162188 | **20650** | 155224 |
| GenHive | 67475 | 13353 | 167297 | 20718 | 147960 |
| PHUNTER | 64717 | **12290** | 146944 | **20650** | 148658 |
| ACO-HH | 73348 | 14371 | 149672 | 21663 | 151610 |
| **HAHA** | 65498 | 13317 | 155941 | 20654 | 148655 |
| ISEA | 70471 | 13339 | 149149 | 20657 | 150474 |
| GISS | 61580 | 13352 | 162266 | 20657 | 149590 |
| SelfS | 73894 | 14386 | 203667 | 20687 | 153590 |
| XCJ | 63654 | 13354 | 152321 | 20658 | 153110 |
| AVEGNep | 77884 | 12397 | 184710 | 20655 | 166742 |
| MCHH-S | 72005 | 13534 | 207891 | 20850 | 160303 |
| HAEA | **60608** | 13342 | 146951 | 20655 | 147283 |
| Ant-Q | 76678 | 14382 | 193827 | 21656 | 160684 |
| NAHH | 65398 | 13358 | 157242 | 20654 | 152081 |

**Table 5.6:** Vehicle routing problem results

was executed without improving the fitness of the solution. The heuristic selection process itself chooses a random heuristic from all available heuristics with the largest rank.

After the local search phase, the resulting solution is compared to the input solution of the preceding shaking step. If the new solution is worse, then the mutation or ruin-recreate heuristic is added to a tabu list. If the new solution has equal fitness then the corresponding mutation or ruin-recreate heuristic is added to the tabu list with a probability of 0.2.

In a subsequent environmental selection step, one of the solutions of the archive is replaced by the new solution. If the shaking and the local search managed to improve the input solution, then it is replaced by this better output. If however there was no improvement, then a 2-tournament selection process on the solution archive is performed in order to find a worse solution to be replaced.

As final ingredient, a periodical adjustment step is performed every 1/10th of the total time budget. In this step, the number of consecutive non-improving iterations allowed for the local search step and the size of the population archive is updated depending on various conditions of the search process.

The VNS-TW hyperheuristic algorithm achieved the second place in the final rank-

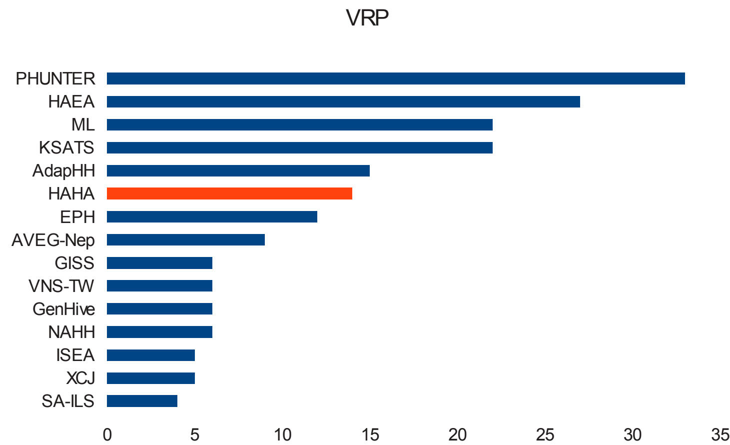


**Figure 5.6:** Vehicle routing problem results

| Rank | Algorithm | Score | Author/Team | Affiliation |
|---|---|---|---|---|
| 1 | AdapHH | 181 | Mustafa Misir | University KaHo Sint-Lieven, Belgium |
| 2 | VNS-TW | 134 | Ping-Che Hsiao | National Taiwan University, Taiwan |
| 3 | ML | 131.5 | Mathieu Larose | Université de Montréa,Canada |
| 4 | PHUNTER | 93.25 | Fan Xue | Hong Kong Polytechnic U., Hong Kong |
| 5 | EPH | 89.75 | David Meignan | Polytechnique Montréal, Canada |
| **6** | **HAHA** | **75.75** | **Andreas Lehrbaum** | **Vienna University of Technology, Austria** |
| 7 | NAHH | 75 | Franco Mascia | Université Libre de Bruxelles, Belgium |
| 8 | ISEA | 71 | Jiri Kubalik | Czech Technical University, Czech Rep. |
| 9 | KSATS-HH | 66.5 | Kevin Sim | Edinburgh Napier University, UK |
| 10 | HAEA | 53.5 | Jonatan Gomez | Univ. Nacional de Colombia, Colombia |
| 11 | ACO-HH | 39 | José Luis Núñez | Universidad de Santiago de Chile |
| 12 | GenHive | 36.5 | CS-PUT | Poznan University of Technology, Poland |
| 13 | DynILS | 27 | Mark Johnston | Victoria University of Wellington, New Zealand |
| 14 | SA-ILS | 24.25 | He Jiang | Dalian University of Technology, China |
| 15 | XCJ | 22.5 | Kamran Shafi | University of New South Wales, Australia |
| 16 | AVEG-Nep | 21 | Tommaso Urli | University of Udine, Italy |
| 17 | GISS | 16.75 | Alberto Acuña | University of Santiago de Chile, Chile |
| 18 | SelfSearch | 7 | Jawad Elomari | Warwick University, UK |
| 19 | MCHH-S | 4.75 | Kent McClymont | University of Exeter, UK |
| 20 | Ant-Q | 0 | Imen Khamassi | University of Tunisia, Tunisia |

**Table 5.7:** Final competition ranking

ing of the competition. The good performance in the Max-SAT and the personnel scheduling classes as well as the rather poor results in the bin-packing domain are similar to our algorithm, indicating some common strengths and weaknesses.

### ML

With his algorithm ML, Larose uses a reinforcement-based variant of iterated local search similar to the method described in [73]. The algorithm repeatedly executes

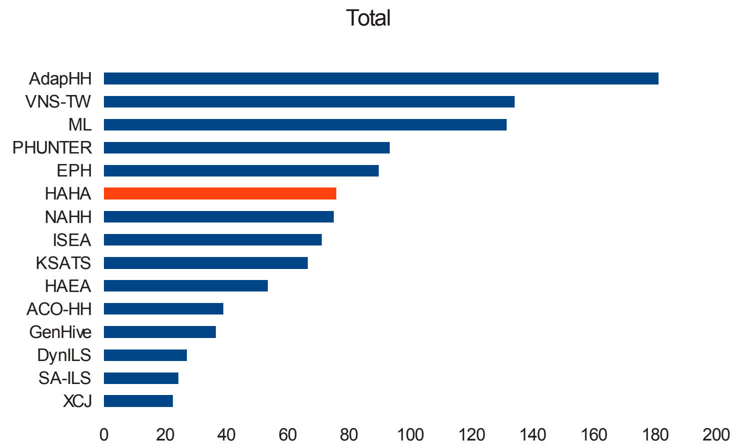


**Figure 5.7:** Total results

Diversification-Intensification cycles, where a solution is first perturbed and then improved as much as possible with the local search heuristics. In the diversification step, ML either applies one of the mutation and ruin-recreate heuristics, or a special no-op heuristic that does not change the solution and therefore gives the subsequent intensification step a second chance to improve the solution. The algorithm uses a move acceptance criterion that only accepts a new solution if it is an improvement over the current solution or if the current solution has not been improved for the last 120 iterations.

For the heuristic selection process, the algorithm performs an Adaptive Large Neighbourhood Search (ALNS) as described in [92]. A set of rules consisting of condition-action pairs is used, where the action corresponds to the available low-level heuristics and the conditions are based on the previously applied heuristics. A continuously updated weight matrix facilitates a reinforcement learning process that tries to increase the weights of beneficial heuristics. A roulette-wheel selection process is performed to choose a heuristic according to its relative weight entry in the matrix.

Not much else is known about the details of the ML algorithm, but it produced very good results among all problem classes, achieving the third place in the overall ranking. One interesting thing to note about the results of ML is, that it was consistently ranked among the top seven algorithms for each problem class. This level of robustness is not even matched by the otherwise outstanding AdapHH algorithm (with a worst-case ranking of 10), giving rise to the implication that reinforcement learning is indeed a very promising area of research for cross-domain hyperheuristics.

### Pearl Hunter

Another very interesting approach is found with Pearl Hunter from Xue which achieved the 4th best result from the competition. Pearl Hunter (PHunter) uses the analogy of a rational diver trying to collect pearls from oysters. Two different modes of dives are used, namely ”snorkeling” (local search heuristics with a low depth of search parameter) and ”deep dive” (local search heuristics with a high depth of search parameter). A common search pattern is to start with a number of initial solutions (positions) where a few snorkeling dives are performed which in turn identify positions worthy of a subsequent deep dive.

In addition to that general structure, four distinct modes of the algorithm are defined: average calls, crossover emphasised, crossover only and average calls with online pruning. These modes are switched according to a decision tree model which uses various success percentages of previous dives as classificiation criteria. The decision tree was trained with the machine learning toolkit WEKA [50] in a separate offline phase.

Pearl Hunter achieved top scores in the vehicle routing problem domain and average to good results in the remaining domains. Because of the fact that it did not have a domain with very bad results, it seems to be a rather robust approach as well. One interesting fact is that Pearl Hunter got the best combined scores on the two hidden domains (TSP and VRP) out of all the participating algorithms. This indicates that the trained decision tree model managed to generalise well on the unseen problem classes.

### Additional Approaches

A number of different additional approaches was employed by the remaining competitors with varying degree of success. EPH from Meignan uses a co-evolutionary approach where a population of solutions is evolved by applying sequences of heuristics which are in turn evolved according to their performance by a separate evolutionary algorithm. Kubalik with his candidate ISEA utilises an evolutionary iterated local search algorithm called POEMS[66] that was demonstrated to perform well on a number of other discrete optimisation problems. Also a hybrid evolutionary approach was used by Gómez with his algorithm HAEA[48].

Ant colony optimisation was used with relatively little success by Núñez with ACO-HH and Khamassi with Ant-Q[62]. A Genetic Hive Hyperheuristic (GenHive) was implemented by Frankiewicz et al., where the analogy of a bee hive is used to describe an evolutionary multi-agent system where each agent employs an evolved sequence of low-level heuristics.

Johnston et al. uses a dynamic iterated local search technique termed DynILS, which applies an adaptive concept of different kick strategies (as described in [68]. These kick strategies are switched according to a reward system that gives advantage to kick moves and strengths that performed well in the past. In the context of the given

task, an individual kick refers to any one of the mutation and ruin-recreate heuristics in combination with the intensity of mutation parameter.

AVEG-Nep[43] by Urli and Di Gaspero uses a reinforcement learning approach with a state formulation that captures the recent trend of the search process. The actions within this reinforcement learning process are not individual heuristics, but heuristic families (local search, crossover, mutation or ruin-recreate) in combination with a parameter (intensity of mutation and depth of search). GISS by Acuna et al. and KSATS-HH by Sim work with a simulated annealing process, the latter one also including a reinforcement learning technique.

# CHAPTER 6

# Conclusion

In this thesis we gave an overview of existing research on hyperheuristics and presented various definitions and classifications. We proposed a new hyperheuristic algorithm for solving cross-domain search problems which uses a straightforward approach of intensification/diversification phases and a novel, quality-based ranking mechanism for low-level heuristics. We tested the algorithm using benchmark instances from six different, well-known combinatorial optimisation problems. The results of the algorithm showed a good general problem solving capability and also revealed potential for further improvements.

We documented the implementation of the algorithm on basis of the HyFlex framework and discussed the results of our submitted candidate for the Cross-domain Heuristic Search Challenge 2011. The statistical weaknesses of the scoring method employed by the challenge are explained and a possible remedy is proposed. We also analysed the competing hyperheuristic approaches, described the best-performing algorithms from the competition and gave an overview about the techniques used by the remaining participants.

## 6.1 Further Research

The ongoing research concerning this algorithm tries to incorporate ideas from a multi-agent reinforcement learning approach (as used in [43]), that applies a completely orthogonal metric for deciding which heuristics to choose. This new approach does not distinguish between the individual low-level heuristics of a certain type, but instead treats them as a single family and only switches between these families of heuristics. We hope to be able to combine the best properties of these two approaches in order to gain additional robustness for different problem domains and to hopefully be successful in future hyperheuristic competitions.

Another issue that is going to be addressed in future versions of the algorithm is the employed quality measure for the local search heuristics, as the competition results indicated that it has potential for improvement in certain problem domains. Some results in the literature also give reason to expect that the usage of different acceptance criteria might turn out to give our algorithm an additional performance improvement.

# Bibliography

[1] U. Aickelin, E.K. Burke, and J. Li. An evolutionary squeaky wheel optimization approach to personnel scheduling. *Evolutionary Computation, IEEE Transactions on*, 13(2):433–443, 2009.

[2] S. Allen, E.K. Burke, M. Hyde, and G. Kendall. Evolving reusable 3d packing heuristics with genetic programming. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 931–938. ACM, 2009.

[3] J. Argelich, L. Chu-Min, F. Manya, and J. Planes. Maxsat evaluation 2009 benchmark data sets. `http://www.maxsat.udl.cat`, 2009.

[4] H. Asmuni, E.K. Burke, and J.M. Garibaldi. Fuzzy multiple heuristic ordering for course timetabling. In *Proc. 5th UK Workshop Comput. Intell*, pages 302–309. Citeseer, 2005.

[5] M. Bader-El-Den and R. Poli. Generating sat local-search heuristics using a gp hyper-heuristic framework. In *Artificial Evolution*, pages 37–49. Springer, 2008.

[6] R. Bai, J. Blazewicz, E.K. Burke, G. Kendall, and B. McCollum. A simulated annealing hyper-heuristic methodology for flexible decision support. *School of CSiT, University of Nottingham, UK, Tech. Rep*, 2007.

[7] R. Baldacci, N. Christofides, and A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385, 2008.

[8] B. Bilgin, E. Özcan, and E.E. Korkmaz. An experimental study on hyper-heuristics and exam timetabling. In *Proceedings of the 6th international conference on Practice and theory of automated timetabling VI*, pages 394–412. Springer-Verlag, 2006.

[9] C. BLUM and A. ROLI. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[10] E. Burke, T. Curtois, M. Hyde, G. Ochoa, and J. A. Vazquez-Rodriguez. HyFlex: A Benchmark Framework for Cross-domain Heuristic Search. *ArXiv e-prints*, July 2011.

[11] E.K. Burke and T. Curtois. Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[12] E.K. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, and J. Antonio. HyFlex: A Flexible Framework for the Design and Analysis of Hyper-heuristics. In *Multidisciplinary International Scheduling Conference (MISTA 2009), Dublin, Ireland*, page 790, 2009.

[13] E.K. Burke, T. Curtois, R. Qu, and G.V. Berghe. A time predefined variable depth search for nurse rostering. *INFORMS Journal on Computing*, 2007.

[14] E.K. Burke, M. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. *Parallel Problem Solving from Nature-PPSN IX*, pages 860–869, 2006.

[15] E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. A survey of hyper-heuristics. *Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham*, 2009.

[16] E.K. Burke, MR Hyde, G. Kendall, and J. Woodward. The scalability of evolved on line bin packing heuristics. In *2007 IEEE Congress on Evolutionary Computation*, pages 2530–2537. Citeseer, 2007.

[17] E.K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[18] E.K. Burke, S. Petrovic, and R. Qu. Case-based heuristic selection for timetabling problems. *Journal of Scheduling*, 9(2):115–132, 2006.

[19] E.K. Burker, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J.R. Woodward. A classification of hyper-heuristic approaches. *Handbook of Metaheuristics*, pages 449–468, 2010.

[20] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.

[21] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[22] J.F. Cordeau, M. Gendreau, G. Laporte, J.Y. Potvin, and F. Semet. A guide to vehicle routing heuristics. *Journal of the Operational Research Society*, pages 512–522, 2002.

[23] P. Cowling and K. Chakhlevitch. Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 2, pages 1214–1221. IEEE, 2003.

[24] P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *wcci*, pages 1185–1190. IEEE, 2002.

[25] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. *Practice and Theory of Automated Timetabling III*, pages 176–190, 2001.

[26] P. Cowling, G. Kendall, and E. Soubeiga. A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the 4th metaheuristic international conference*, volume 1101, pages 127–131. Citeseer, 2001.

[27] CRIL. Sat competition 2007 benchmark data sets. centre de recherche en informatique de lens. `http://www.cril.univ-artois.fr/SAT07`, 2007.

[28] CRIL. Sat competition 2009 benchmark data sets. centre de recherche en informatique de lens. `http://www.cril.univ-artois.fr/SAT09`, 2009.

[29] T. Curtois. Staff rostering benchmark data sets. `http:///www.cs.nott.ac.uk/~tec/NRP/`, 2009.

[30] T. Curtois, G. Ochoa, M. Hyde, and J.A. Vázquez-Rodríguez. A hyflex module for the personnel scheduling problem. *School of Computer Science, University of Nottingham, Tech. Rep*, 2009.

[31] G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management science*, pages 80–91, 1959.

[32] J. Denzinger, M. Fuchs, and M. Fuchs. High performance atp systems by combining several ai methods. In *Proceedings of the 15th international joint conference on Artifical intelligence-Volume 1*, pages 102–107. Morgan Kaufmann Publishers Inc., 1997.

[33] M. Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.

[34] K.A. Dowsland, E. Soubeiga, and E. Burke. A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. *European Journal of Operational Research*, 179(3):759–774, 2007.

[35] J. Dréo and C. Candan. Metaheuristics classification. `http://upload.wikimedia.org/wikipedia/commons/c/c3/Metaheuristics_classification.svg`, 2011.

[36] ESICUP. European special interest group on cutting and packing benchmark data sets. `http://paginas.fe.up.pt/~esicup`, 2011.

[37] H. Fisher and G.L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial scheduling*, pages 225–251, 1963.

[38] A.S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):31–61, 2008.

[39] M.R. Garey and D.S. Johnson. *Computers and intractability*, volume 174. Freeman San Francisco, CA, 1979.

[40] P. Garrido and C. Castro. Stable solving of cvrps using hyperheuristics. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 255–262. ACM, 2009.

[41] P. Garrido and M.C. Riff. Dvrp: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *Journal of Heuristics*, 16(6):795–834, 2010.

[42] L. Di Gaspero and A. Schaerf. Easysyn++: A tool for automatic synthesis of stochastic local search algorithms. *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*, pages 177–181, 2007.

[43] L. Di Gaspero and T. Urli. A reinforcement learning approach for the cross-domain heuristic search challenge. 2011.

[44] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, pages 1086–1094, 1992.

[45] I.P. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for sat. In *Proceedings of the national conference on artificial intelligence*, pages 28–28. Citeseer, 1993.

[46] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[47] F. Glover et al. Tabu search-part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[48] J. Gomez. Self adaptation of operator rates in evolutionary algorithms. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 1162–1173. Springer, 2004.

[49] J. Gratch, S. Chien, and G. DeJong. Learning search control knowledge for deep space network scheduling. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 135–142. Citeseer, 1993.

[50] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[51] L. Han and G. Kendall. Guided operators for a hyper-heuristic genetic algorithm. *AI 2003: Advances in Artificial Intelligence*, pages 807–820, 2003.

[52] J.H. Holland. Adaptation in natural and artificial systems. an introductory analysis with applications to biology, control and artificial intelligence. *Ann Arbor: University of Michigan Press, 1975*, 1, 1975.

[53] M. Hyde and G. Ochoa. Hyflex competition instance summary. 2011.

[54] M. Hyde, G. Ochoa, T. Curtois, and JA Vázquez-Rodríguez. A hyflex module for the boolean satisfiability problem. Technical report, Technical report, School of Computer Science, University of Nottingham, 2009.

[55] M. Hyde, G. Ochoa, T. Curtois, and JA Vázquez-Rodríguez. A hyflex module for the one dimensional bin-packing problem. *School of Computer Science, University of Nottingham, Tech. Rep*, 2009.

[56] M. R Hyde. One dimensional packing benchmark data sets. `http://www.cs.nott.ac.uk/~mvh/packingresources.shtml`, 2011.

[57] A. Ikegami and A. Niwa. A subproblem-centric model and approach to the nurse scheduling problem. *Mathematical Programming*, 97(3):517–541, 2003.

[58] D.E. Joslin and D.P. Clements. Squeaky wheel optimization. In *Proceedings of the national conference on artificial intelligence*, pages 340–346. John Wiley & Sons Ltd, 1998.

[59] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Arxiv preprint cs/9605103*, 1996.

[60] RE Keller and R. Poli. Linear genetic programming of parsimonious metaheuristics. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 4508–4515. IEEE, 2007.

[61] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948. IEEE, 1995.

[62] I. Khamassi, M. Hammami, and K. Ghedira. Ant-q hyper-heuristic approach for solving 2-dimensional cutting stock problem. In *Swarm Intelligence (SIS), 2011 IEEE Symposium on*, pages 1–7. IEEE, 2011.

[63] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671, 1983.

[64] R K. KotovskyJ and HA Simon. Why are some problems hard? evidence from tower of hanoi. *Cognitive psychology*, 17(2):248–294, 1985.

[65] J.R. Koza. Genetic programming. 1992.

[66] J. Kubalik and J. Faigl. Iterative prototype optimisation with evolved improvement steps. *Genetic Programming*, pages 154–165, 2006.

[67] R. Kumar, A.H. Joshi, K.K. Banka, and P.I. Rockett. Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1227–1234. ACM, 2008.

[68] T. Liddle. Kick strength and online sampling for iterated local search. In *Proceedings of the 45th Annual Conference of the ORSNZ*, 2010.

[69] H. Lourenço, O. Martin, and T. Stützle. Iterated local search. *Handbook of metaheuristics*, pages 320–353, 2003.

[70] J. Marín-Blázquez and S. Schulenburg. A hyper-heuristic framework with xcs: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. *Learning classifier systems*, pages 193–218, 2007.

[71] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[72] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proceedings of the national conference on artificial intelligence*, pages 321–326. John Wiley & Sons Ltd, 1997.

[73] D. Meignan, A. Koukam, and J.C. Créput. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879, 2010.

[74] M. Misir, P. De Causmaecker, G. Vanden Berghe, and K. Verbeeck. An adaptive hyper-heuristic for chesc 2011. *status: published*, 2011.

[75] A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. *Applied Optimization*, 86:523–544, 2003.

[76] M. Nawaz, E.E. Enscore Jr, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[77] A. Newell, J.C. Shaw, and H.A. Simon. Report on a general problem-solving program. In *Proceedings of the international conference on information processing*, volume 1, pages 256–264, 1959.

[78] I.P. Norenkov and E.D. Goodman. Solving scheduling problems via evolutionary methods for rule sequence optimization. *Soft computing in engineering design and manufacturing*, page 350, 1998.

[79] M. Oltean. Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, 13(3):387–410, 2005.

[80] M. Oltean and D. Dumitrescu. Evolving tsp heuristics using multi expression programming. *Computational Science-ICCS 2004*, pages 670–673, 2004.

[81] M. Oltean and C. Grosan. Evolving evolutionary algorithms using multi expression programming. *Advances in Artificial Life*, pages 651–658, 2003.

[82] I. Or. *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*. PhD thesis, Northwestern University, 1976.

[83] D. Ouelhadj and S. Petrovic. A cooperative distributed hyper-heuristic framework for scheduling. In *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*, pages 2560–2565. IEEE, 2008.

[84] E. Özcan, B. Bilgin, and E.E. Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.

[85] N. Pillay and W. Banzhaf. A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem. *European Journal of Operational Research*, 197(2):482–491, 2009.

[86] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.

[87] R. Poli, J. Woodward, and E.K. Burke. A histogram-matching approach to the evolution of bin-packing strategies. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3500–3507. IEEE, 2007.

[88] I. Rechenberg. Evolutionsstrategie–optimierung technischer systeme nach prinzipien der biologischen evolution. 1973.

[89] G. Reinelt. Tsplib - a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[90] G. Reinelt. Tsplib, a library of sample instances for the tsp. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95`, 2008.

[91] J.R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[92] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[93] P. Ross and JG Marfn-Blazquez. Constructive hyper-heuristics in class timetabling. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1493–1500. IEEE, 2005.

[94] P. Ross, S. Schulenburg, J.G. Marín-Blázquez, and E. Hart. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2002, pages 942–948. Citeseer, 2002.

[95] B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the national conference on artificial intelligence*, pages 337–337. John Wiley & Sons Ltd, 1994.

[96] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the tenth national conference on Artificial intelligence*, pages 440–446. Citeseer, 1992.

[97] SINTEF. Vrptw benchmark problems. `http://www.sintef.no/Projectweb/TOP/Problems/VRPTW`, 2011.

[98] R.H. Storer, S.D. Wu, and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management science*, pages 1495–1509, 1992.

[99] E. Taillard. Flow shop benchmark data sets. `http://mistic.heig-vd.ch/taillard`, 2010.

[100] J.C. Tay and N.B. Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453–473, 2008.

[101] H. Terashima. Generalized hyper-heuristics for solving 2D Regular and Irregular Packing Problems. *Annals of Operations Research*, pages 1–24, 2010.

[102] H. Terashima-Marin, EJ Flores-Alvarez, and P. Ross. Hyper-heuristics and classifier systems for solving 2d-regular cutting stock problems. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 637–643. ACM, 2005.

[103] H. Terashima-Marin, CJ Farias Zarate, P. Ross, and M. Valenzuela-Rendon. Comparing two models to generate hyper-heuristics for the 2d-regular bin-packing problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 2182–2189. ACM, 2007.

[104] J.A. Vázquez-Rodrıguez, G. Ochoa, T. Curtois, and M. Hyde. A hyflex module for the permutation flow shop problem. *School of Computer Science, University of Nottingham, Tech. Rep*, 2009.

[105] JA Vazquez-Rodriguez, S. Petrovic, and A. Salhi. A combined meta-heuristic with hyper-heuristic approach to the scheduling of the hybrid flow shop with sequence dependent setup times and uniform machines. In *Proceedings of the 3rd Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2007)*, pages 506–513, 2007.

[106] B.W. Wah and A. Ieumwananonthachai. Teacher: A genetics-based system for learning and for generalizing heuristics. *Evolutionary Computation. World Scientific Publishing Co. Pte. Ltd*, 1998.

[107] B.W. Wah, A. Ieumwananonthachai, L.C. Chu, and A.N. Aizawa. Genetics-based learning of new heuristics: rational scheduling of experiments and generalization. *Knowledge and Data Engineering, IEEE Transactions on*, 7(5):763–785, 1995.

[108] S.W. Wilson. Classifier fitness based on accuracy. *Evolutionary computation*, 3(2):149–175, 1995.

[109] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.