JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
**Andreas Beham**

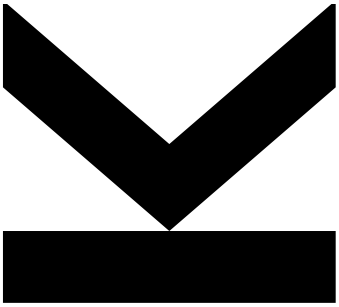
Submitted at
**Institute for Formal Models and Verification**

Supervisor and
First Examiner
**Michael Affenzeller**

Second Examiner
**Enrique Alba**

June 2019

# Fitness Landscape Analysis and Algorithm Selection for Assignment Problems



Doctoral Thesis
to obtain the academic degree of
Doktor der technischen Wissenschaften
in the Doctoral Program
Technische Wissenschaften

**JOHANNES KEPLER UNIVERSITY LINZ**
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

# Acknowledgments

This thesis is the result of a long endeavor in the application of metaheuristic optimization in real-world projects. As such I have to thank many people that stood by me through the years. I have to thank my family, my colleagues, and the fellow scientists that are adventuring in the same waters. I thank my life companion Dona D. and my son Vincent E. for all the fun we have together and for enduring most of the long working hours. My father Wolfgang, my brother Werner, my sister Marielis, and grand parents who support and stand by me. My boss, supervisor, mentor, and colleague Michael Affenzeller who I thank for his hard work and for the spirit with which he directs our research group. My colleagues Stefan, Gabriel, Michael, Stephan, Erik, Johannes, Sebastian, Viktoria, Philipp, Bogdan and the rest of the research group as well as former colleagues Roland, Andreas and Judith have to be thanked for the great time that we spent together. I want to thank the FH Oberösterreich which is a thriving institution and a great place to work at. I want to thank the Johannes Kepler University for creating high quality study programmes and its dedicated professors Armin Biere and Gü for their support. I want to thank my second supervisor Enrique Alba especially, for his hard work dedicated to the advancement of our scientific field and his continued support for young researchers. Last but not least, I want to thank the hard working people at our company partners without whom successful applications and achieving real-world results in our projects, and as a consequence also this thesis, would have been entirely impossible.

# Abstract

The field of optimization has a wide range of applications in the real world. The operation of trucks, ships, cranes, production lines, hospitals, or warehouses results in many decision situations that affect efficiencies, business capabilities, and costs. Deterministic and stochastic models formalize the decision situations as well as their objectives. Exact and heuristic techniques are employed to solve these models efficiently. The resulting solutions are then used to derive the respective decisions. A large range of such solving algorithms are available, but which outperform each other in different situations. It is thus a major problem to decide which algorithm instance should be applied to solve a specific model.

It is the goal of this thesis to devise and evaluate methods that are able to solve this so called algorithm selection problem. We will evaluate whether an improved solving algorithm can be found that reuses existing approaches. The goal is to evaluate such an algorithm when applied to well-known models and evaluate its performance. In this light, research and development efforts from three domains are described and studied.

First, fitness landscape analysis is a method applicable to characterize models and their instances through a set of features. We will provide an overview on high-level characteristics that influence the performance of solving algorithms. In this thesis a new method termed directed walk and new features are devised, different variants thereof are analyzed, and new ideas for algorithms are given.

Second, the algorithm selection problem is solved in two case studies. Several thousand runs have been performed on a wide range of different benchmark instances. Thereby, the performance of a range of algorithms is recorded. In these two studies we will evaluate a classification-based approach to the algorithm selection problem. Thereby, we will compare the performances of algorithms and relate those to the landscape characteristics. We will show that a combined approach performs better than an individual algorithm.

Third, the general research model of operations research is introduced and it is discussed how such results can be transferred to other models and domains. Thereby, we discuss the steps involved to move from a real-world problem to a scientific model and further to a solution through an efficient solver. Finally, we introduce software systems that support such tasks and which have been created by the author in the course of his studies.

# Kurzfassung

Optimierungsalgorithmen finden vielfach Anwendung bei Planungs- und Steuerungsaufgaben. Im Betrieb von Lastkraftwagen, Shiffen, Kränen, Produktionslinien, Spitälern oder Lagerhäuser sind Entscheidungen zu treffen, die die Effizienz, Handlungsmöglichkeiten und Kosten beeinflussen. Deterministische und stochastische Modelle formalisieren diese Entscheidungssituationen und deren Ziele. Exakte und heuristische Ansätze werden zur effizienten Lösung solcher Modelle eingesetzt und entsprechende Entscheidungen werden aus deren Lösungen abgeleitet. Eine Vielzahl unterschiedlicher Algorithmen zur Lösungsfindung sind verfügbar, übertreffen sich jedoch gegenseitig bei unterschiedlichen Problemen. Es ist daher entscheidend den richtigen Algorithmus auszuwählen.

Ziel dieser Arbeit ist es, Methoden die das sogenannte Algorithmus Auswahlproblem lösen, zu entwickeln und zu evaluieren. Wir werden verbesserte Methoden auf Basis bestehender Algorithmen entwicklen und in der Anwendung auf bekannte Modelle untersuchen. In dieser Hinsicht werden Forschungs- und Entwicklungstätigkeiten aus drei Bereichen zusammengeführt.

Zuerst wird das Gebiet der Fitnesslandschaftsanalyse vorgestellt und wie damit Modelleinstanzen durch Merkmale beschrieben werden. Nach einem Überblick über Landschaftsaspekte und deren Einflüsse auf das Leistungsverhalten von Algorithmen, werden neue Methoden und Merkmale vorgestellt und analysiert. Aufbauend darauf werden Ideen für neue Algorithmen diskutiert.

Im zweiten Teil der Arbeit wird das Algorithmus Auswahlproblem in zwei Fallstudien gelöst. Mehrere tausend Versuche wurden hierfür auf einer breiten Palette von Modellinstanzen durchgeführt. Aus dem beobachteten Konvergenzverhalten vieler Algorithmen wird in den beiden Fallbeispielen ein klassifikationsbasierter Ansatz zur Lösung des Auswahlproblems evaluiert. Die Resultate aus den Versuchen werden vergleichend gegenübergestellt und Zusammenhänge mit Landschaftseigenschaften untersucht. Wir werden zeigen, dass ein kombinierter Ansatz in der Lage ist einzelne Algorithmen zu übertreffen.

Im dritten Teil der Arbeit wird, anhand einem allgemeinen Vorgehensmodell, diskutiert wie sich dies auf andere Modelle und Domänen übertragen lässt. Dabei werden Entscheidungen beschrieben, die für die Überführung einer echten Entscheidungssituation in ein wissenschaftliches Modell notwendig sind. Zum Schluß werden Softwaresysteme vorgestellt, die solche Entscheidungen unterstützen und die vom Autor während seines Studiums erstellt wurden.

# Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die vorliegende Dissertation ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, Juni 2019

Andreas Beham

# Contents

# List of Figures

# List of Tables

# List of Algorithms



# List of Code Listings

# 1 Introduction

> "The step from the plus to the comma version finally seemed to climb to the top of mount nonsense [..]"
>
> H-G. Beyer and H-P. Schwefel [BS02]

Heuristic problem solving is essential in many fields of application. From finding a path between A and B, deciding on the next of a number of tasks, selecting several from a range of options, or identifying patterns in data. There are optimization problems at the heart of many decision situations.

Determining good facility locations is a central problem when a business is expanding or new businesses are formed. There are multiple approaches how to model and solve such problems. The capacitated plant location problem [Sri95] describes the minimization of costs when depots are opened in a set of possible locations to serve a range of customers. Two types of costs exist, on the one hand, opening a depot at a location creates installation costs. On the other hand, service costs arise when customers are served. In such a case where facilities exchange goods with each other, assignment problems may be used to model the corresponding decisions [KB57, RS75, PHGZ10].

The vehicle routing problem is a core problem in truck dispatching scenarios [TV14]. It describes a minimization of resources, e.g. cost, time, or distance, of several vehicles' tours, where each vehicle usually starts from and returns to the same depot after servicing a range of sites. Advanced applications such as the location routing problem combine the choice of tours with the choice of where to put the depots as introduced just before [PP14]. Inventory routing problems combine an optimization of tours with the additional decision on determining which goods are to be serviced, thus managing the inventory of the sites [CCL13, VAB+13]. There is a rich literature of potential applications in many areas such as health care [SDH11], intralogistics [VBK+13], as well as solution approaches [AD04, VBA+12, VCGP14].

Scheduling problems are at the heart of planning operations in manufacturing or service industry [Pin16]. Scheduling describes a minimization, for instance of the completion time or tardiness of a range of activities. The activities require certain renewable or non-renewable resources while they are being

executed. Additionally, there may be dependencies to other activities that must have been completed earlier [HB10]. Applications for scheduling problems arise in health care [BDBV04], production [VW07, HKB+18, HBR+19], or in computing environments [NCA11].

These and many other problems are in the class of *NP-hard* problems, meaning that there is no polynomial time algorithm that guarantees finding the optimal solution. The exponential growth of the problems' complexity creates barriers for solving larger and larger instances and problem specific heuristics may need to be created. For example, to identify a good round-trip between a number of cities one could use a heuristic to choose a starting city, e.g. the one that is farthest away from all others and then use a heuristic to choose a connecting city, e.g. the one that is closest. A round trip can then be created by applying these heuristics until all cities are connected. The heuristics include problem specific parameters (i.e. distance) in their decision making. Path problems that are not described by a distance (alone) cannot be solved by these heuristics. For simple problems, such heuristics may already perform very well. However, problems, especially in the real world, can become increasingly complex and sometimes it can be a challenge to find a *feasible* solution.

*Meta*heuristics on the other hand are not specific to a certain problem, they make use of heuristics in that they apply them according to a certain strategy. Often these heuristics are then called *operators*. Genetic algorithms (GA) are a family of metaheuristics that use the heuristic operations *crossover*, *mutation*, and *selection* in order to update a population of solutions [Hol75, Mic99]. GAs also describe what these heuristics should actually do and specify some requirements, e.g. "crossover should take two parent solutions and produce a new child solution that consists, at best, only of parts that are present in the given parent solutions". However, not all metaheuristics may be described in such a general way. Some methods depend on a specific solution encoding. For instance, earlier evolution strategies (ES) by Rechenberg and Schwefel [Rec73, BS02] still describe a general concept of adapting the *strength* of a mutation over the course of the search. Newer variants of ES such as CMA-ES [HO01] specify a detailed adaption strategy, but restricted to the solution space of real-valued vectors. Such metaheuristics can be described as being at least "encoding specific", because they still work for a large range of (real-valued) problems. A more in-depth discussion on metaheuristics is given in Section 2.2.

Many heuristic operators such as the ones mentioned above have been created, some work very well, some work very fast, some work only on some problems, and others work only in combination with certain other heuristic operators. The introduction of metaheuristic algorithms has mitigated, but not solved the problem of identifying "the right" heuristic operator. In addition, while we have highlighted that metaheuristics are more generally applicable, we have to state, that the performance of metaheuristics is again highly problem dependent. A metaheuristic that achieves good performance on routing problems may perform poorly on assignment problems. It can be observed, as also shown later in this thesis, that performance varies even between problem instances, i.e. different parameterizations of a problem. Related to these observations is the theorem that has been formulated by David Wolpert and William Macready and which has been termed "no free lunch" theorem [WM97]. In this well-known work, Wolpert and Macready state that the average performance of any algorithm over all problems is equal.

In the light of this background, the scientific community has started to work on new approaches that attempt to identify the concrete subsets for which certain algorithms are better suited than others. These approaches try to identify patterns in the way algorithms work and patterns in the way the problem instances are shaped. By "overlapping" these two patterns it is hoped to capture what human optimization experts describe as *experience* when they attempt to choose suitable methods and parameters. In identifying a relationship among problems, among algorithms, and between problems and algorithms new insight shall be gained that will potentially also reveal unexplored areas of algorithmic strategies. The foundation for these new developments date back to John R. Rice [Ric76] and the algorithm selection problem (ASP) that will be introduced in more detail later. So far the approaches that are pursued and discussed in the scientific community that attempt to solve the "adaptation problem" of algorithms to problems are:

- Metaoptimization
- Hyperheuristics
- Algorithm Selection

Metaoptimization is the attempt to optimize the parameters of metaheuristics using another metaheuristic [BE12]. The search space of such a metaoptimizer is the parameter configuration space of the underlying metaheuristic. Methods have been described to perform metaoptimization [HHLB11,

BBLP05, HHLBS09]. One well-known software package that performs this task is called *irace* and developed among others by Manuel López-Ibáñez [LIDLSB11]. Metaoptimization is a computationally intensive offline approach and itself subject to the problem of parameterization [DCDS17].

The idea of hyperheuristics is similar to metaoptimization, but focuses on choosing or generating heuristics instead of an adaption of numerical parameters. Hyperheuristics can be applied online or offline [BGH+13]. In an online application the hyperheuristic chooses or generates heuristics during the course of the search, while in an offline application heuristics are chosen or generated on a set of training instances. For instance, the genetic programming approach that the author has presented at IPDPS in 2008 may be viewed as an offline hyperheuristic approach [BWWA08].

Algorithm selection and hyperheuristics are also closely linked together. The problem of choosing among a set of heuristics those that should be applied can be seen as solving an algorithm selection problem. An approach based on the ASP is to memorize the performance data of past experiments, e.g. in the form of a database, and identify a best match with a previously observed case to decide on a new and unseen situation [BAW17]. This may be less useful during the course of the search, but rather before the start of the search.

Summarizing these developments, the field of heuristic optimization has created a large range of different methods with different levels of abstractions and performance. The thesis at hand describes research efforts in the category of algorithm selection based approaches. It is stated how fitness landscape analysis (FLA) can be seen as a method for identifying problem characteristics and describes and evaluates machine learning approaches for pattern identification. Finally, an algorithm selection approach is introduced, tested, and the observed results are discussed.

A short note on the wording and the terms that are being used throughout the thesis: Heuristic algorithms and metaheuristics can be summarized as being *approximate* algorithms in contrast to *exact* or *complete* algorithms. The term "heuristic optimization algorithm" may refer to both a simple heuristic, as well as a complex metaheuristic. The term heuristic may be used instead of approximate. Thus, the term heuristic may be used as adjective as well as noun. The term "approximate" has not been picked up in a major proportion of the scientific literature that discusses these approaches.

## 1.1 Motivation and Goals

Since genetic algorithms [Hol75], evolution strategies [Rec73], and simulated annealing [KGV83] have been introduced a plethora of further metaheuristic algorithms has emerged. Today, the scientific community knows about different algorithm such as ant colony optimization [DD99], particle swarm optimization [KE95], tabu search [Glo86], scatter search [Glo99], variable neighborhood search [HM01], large neighborhood search [PR10], parameter-less population pyramid [GP14], PILOT method [VFD05], greedy randomized adaptive search procedure [FR95], non-dominated sorting genetic algorithm [DPA+02], and strength pareto evolutionary algorithm [ZLT01] to name just a few. While a variety of algorithms may be regarded positively, the disadvantage is that one has to choose a concrete implementation of a fully parameterized algorithm for solving a concrete instances of a certain problem. A rigorous mathematical analysis of this methods could reveal several characteristics that would allow making a good choice given a certain problem instance. However, the analysis is complicated by a range of, ironically desired, properties that are often sought in the creation of metaheuristic algorithms:

- Stochasticity increases the robustness of metaheuristics
- Parameters enable to adapt the metaheuristic to differences in the problems
- Metaheuristics are applicable to a wide range of different problems

### 1.1.1 Challenges and Limitations for Metaheuristics Research

After a phase similar to a cambrian explosion ([CM03]) of, in this case, metaheuristic methods, in recent years critical articles have emerged that question whether some of the proposed novel methods are only metaphorically different and based on the same or highly similar underlying principles [Sör15]. Some methods like harmony search [GKL01] have even caused open dispute in the scientific literature [Wey12]. More recently, the intelligent water drop algorithm has been rigorously analyzed and is claimed to be a special case of ant colony optimization which was introduced much earlier [CVDS19]. Still, the scientific community is looking for new problem-solving concepts to further this field and solve real-world problems better and faster than before. Research on portfolio-based approaches that include a range of algorithms and a selector that returns a suitable algorithm instance constitutes a promising

direction for future research. Nevertheless, there are still open challenges that demand further research and to which this thesis may contribute:

1. Feature Design and Landscape Analysis
2. Algorithm Selection
3. Visualization and Analysis

The first challenge is to describe the state of the dynamic system that is comprised by the application of algorithms to solve problems and the resulting performance that is observed. We need a better understanding of that system and what variables influence the dynamic in what way. In this thesis, like in many related work, it is sought to obtain features from static observations, but also from such dynamic processes to describe the system. Section 3 describes directed walks and some landscape characteristics derived from those.

The second challenge is to identify, study, and analyze models that can be used in algorithm selection and the performance of a portfolio in comparison to individual algorithms. These use the features as inputs and aim to predict the performance of a range of algorithms whereupon the best may be selected. Specifically, in Section 4 of this thesis a k-nearest-neighbor model is evaluated in two case studies from the domain of assignment problems.

Additionally, in Sections 3 and 4 we will also explore different ways to analyze and visualize results. We will devise a new visualization for showing the best algorithm instances in the algorithm selection case studies.

However, still there are several general limitations that may not be overcome. One such limitation concerns the amount of dimensions on which algorithms compete with each other [Ric76, BB06]:

1. *efficiency* - concerns the runtime of an algorithm
2. *reliability* - concerns the quality of the achieved solution
3. *robustness* - concerns an algorithm's ability to achieve good quality among a range of problems
4. *simplicity* - concerns the complexity of an algorithm's implementation

It is important to state that not all of these characteristics can be measured on the algorithm itself, but have to be evaluated in the context of a concrete implementation, potentially using a certain framework. So, a genetic algorithm implemented using e.g. HeuristicLab [Wag09, WKB+14] must be compared to a genetic algorithm implemented using e.g. the DEAP framework

[FDG+12] as if they were separate algorithms. The point is that we cannot compare algorithms directly in a way that would lead to conclusions such as "tabu search outperforms genetic algorithms". First we have to acknowledge that *the* genetic algorithm is not a testable entity, but rather a conceptual framework - which is the case for almost all metaheuristics. Second we must acknowledge that we can test only when we have specified all parameters of an algorithm, leading to the comparison between *algorithm instances* rather than between algorithms themselves. Third we recognize that implementations in programming languages of these instances may be different potentially leading to different properties. Fourth we experience that any implementation of an algorithm instance can only be run on certain hardware. For instance, simplicity of an algorithm is a property of the implementation, but efficiency, reliability, and robustness are properties of the implementation being executed on some hardware. These are not challenges that we may overcome in this thesis, but limitations that are accepted in this work.

### 1.1.2 Intended Contribution to the Field of Metaheuristics

For those researchers that seek to solve optimization problems, the current state of research is difficult to comprehend and understand. Each new published method is often stated to be better than the ones it compares against on the problem instances it compares them on. It is hardly possible to compare against all methods due to the vast number of them and the scattered implementations.

The motivation for this thesis roots in the high potential of applying metaheuristic optimization in practice. The trend to apply artificial intelligence (AI) methods in practice is becoming ever stronger. Metaheuristics are a highly promising branch of AI in that they can deliver high quality solutions to real-world problems in short time. Decision scenarios such as in production or logistics planning, and operational decision situations can be supported by metaheuristic optimization. Still, the requirement of human experts in the application of these techniques presents a stumbling block and slows down its adoption in the real world. My motivation is to support the step of choosing a suitable algorithm that can solve a certain problem efficiently.

The goal of this thesis is to develop and evaluate methods with which it is possible to perform automated choices for solving a certain new and unforseen problem instance given a range of different algorithms. The resulting approach

should be applicable in automated decision making (ADM) scenarios where human observation and control is at least partly unavailable. The vision is that we can devise a system that learns over time, much like a human expert, and can make better and better choices on solving a new unknown problem instance with a range of methods that are at its disposal. When an unknown problem instance arrives, we want to be able to quickly determine a good parametrized algorithm in that sense that we can maximize the performance dimensions that we have specified above. In order to do this we have to treat problem instances not as independent from each other, but by having some sort of similarity which correlates with the performance of successful algorithmic approaches.

## 1.2 Relevant Research Projects

The methods and results described in this thesis would not have been possible without the knowledge and expertise acquired in several research projects that I was engaged in. Among these, the Josef Ressel Centre for Heuristic Optimization (Heureka!) and the K-Project Heuristic Optimization in Production and Logistics (HOPL) have to be mentioned. These projects were devoted to both methodological research and real-world applications.

In recent years, I have received grants for research projects, such as "Integrated Methods for Robust Production Planning and Control (SimGenOpt2)" in the $19^{th}$ call "Produktion der Zukunft" (engl. production of the future) and "Digitale Methoden für verbesserte Personalqualifizierungsstrategien (Optimal Workforce)" in the call "Innovatives Oberösterreich 2020: Digitalisierung" (engl. innovative Upper Austria 2020: digitalization). Among other industrial branches the steel industry has been a fruitful ground for optimization. During HOPL we implemented algorithms that optimize the dispatching of cranes, tractors, and straddle carriers. The digitalization of the production industry under the term "Industry 4.0" is an ongoing process and computational intelligence methods are part of this transformation.

In several of the research projects decision making optimization algorithms were created that run unobserved by humans on servers, machines, or other hardware. It is my personal conviction that this trend will continue in the future. In Section 6.2, I will provide an outlook and discuss briefly the ramifications of these developments on the real world and its current limitations.

# 2 Foundations

> "The random number seed is the only random element during the optimization run."
>
> Thomas Bartz-Beielstein [BB06]

In this section, five parts are to be discussed that form the basis of this thesis. It must be noted that algorithm selection requires the consideration of many different aspects within applied optimization. An in-depth discussion of each of the involved parts is out of the scope of this thesis. The broader view and the discussion and presentation of the involved methods and domains should however be a valuable contribution to anyone seeking to continue and extend the work described here.

1. The first part is related to the problems that are to be studied. The problems are introduced together with some practical applications.

2. In the second section heuristic methods are examined more closely. The similarities and differences between metaheuristics are going to be highlighted and well-known algorithms are introduced.

3. The third part is dedicated to fitness landscape analysis (FLA) and how it is used to characterize and relate problems (resp. the induced fitness landscapes) to each other.

4. The fourth part concerns the performance measurement of algorithms. How can we record and represent performance and how can we compare algorithms based on their performances?

5. The fifth part is concerned with algorithm selection problems themselves and the respective solution approaches.

## 2.1 Assignment Problems

Assignment problems are relevant in the real world, because often in our life and in business operations we assign items from one set to items from another. For instance, support tickets are assigned to members of an IT department, medications are put in storage bins of refrigerated warehouses, steel slabs are stacked in open yards, kitchen utensils, herbs, ingredientc, etc. are distributed among drawers and cupboards in our kitchens, employees are given a desk in an office. Thus, the entities are often named workers, facilities, equipments, jobs on the one hand and tasks, locations, and resources on the other hand. In terms of the mathematics the name is irrelevant, but it is important to note that there are two distinct sets of items.

A solution to an assignment problem then defines a relation between items of the first set to items of the second set. In some problem definitions given below the relation has some requirements that it needs to fulfill, e.g. bijective, injective, or surjective constraints. For instance, in case of a bijection each element of set one must be paired with exactly one element of set two and vice versa. Such an assignment solution is expressed in terms of the pairings. For instance, each possible pairing might be described by a variable that is 0 if the pairing is absent and 1 if it is present. Finally, there exists an objective function that maps the solution into the domain of real numbers and which must be minimized or maximized. In the following some well-known definitions of assignment problems are given.

### 2.1.1 Linear Assignment Problem

The linear assignment problem (LAP) is not NP-hard. It can be solved to optimality in polynomial time using the so called Hungarian algorithm [Kuh55]. The problem can be described as assigning $N$ independent workers to $N$ independent tasks. An $N \times N$ cost matrix $C$ with elements $c_{ij}$ describes resources (i.e. time, money) that worker $i$ would consume when completing task $j$. The goal is to find a permutation $\pi$ of the numbers $1, .., N$ with $\pi(i)$ denoting the element at position $i$ so that the following objective is achieved:

$$\min \sum_{i=1}^{N} c_{i\pi(i)}$$

### 2.1.2 Quadratic Assignment Problem

The quadratic assignment problem (QAP) is NP-hard [KB57]. It is an extension to the LAP. In this model $N$ facilities are assigned to $N$ locations such that the facilities that are frequently communicating lie closer together. A weight matrix $W$ with elements $w_{ij}$ describes the strength of the exchange between facilities $i$ and $j$ while the distance matrix $D$ with elements $d_{ts}$ describes the distance between locations $t$ and $s$. The goal is to find a permutation $\pi$ of all elements in the domain $\mathbb{D} = [1; N]$ with $\pi(i)$ denoting the element at position $i$ so that the following objective is achieved:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} \cdot d_{\pi(i)\pi(j)} \tag{2.1}$$

This is the purely quadratic assignment problem, if a linear term is present in form of the cost matrix $C$ with elements $c_{ij}$ describing the cost of installing facility $i$ in location $j$ the objective function becomes

$$\min \sum_{i=1}^{N} \left( c_{i\pi(i)} + \sum_{j=1}^{N} w_{ij} \cdot d_{\pi(i)\pi(j)} \right) \tag{2.2}$$

But it is often difficult to describe the installation costs and the costs arising out of the strength-distance product in terms of a common domain, i.e. currency and thus the simpler objective is the more commonly known and studied one. The QAP, as can be seen in equation (2.1), does not define constraints. It can be written as an integer programming problem, which introduces those constraints that would otherwise be satisfied with $\pi$ being a permutation (in equations (2.4) and (2.5)):

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} \sum_{s=1}^{N} x_{ij} \cdot x_{ks} \cdot w_{ik} \cdot d_{js} \tag{2.3}$$

$$\text{s.t.} \sum_{i=1}^{N} x_{ij} = 1 \quad \forall j \in \{1, ..., N\} \tag{2.4}$$

$$\sum_{j=1}^{N} x_{ij} = 1 \quad \forall i \in \{1, ..., N\} \tag{2.5}$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in \{1, ..., N\} \wedge j \in \{1, ..., M\} \tag{2.6}$$

Another aspect of the quadratic assignment problem is that it generalizes the well-known traveling salesman problem (TSP) [ABCC07]. The path of the salesman can be represented by constructing a special flow matrix, as shown below. This flow of unit strength visits all "facilities" exactly once.

$$\begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

### 2.1.3 Generalized Assignment Problem

The LAP and QAP both consider a "1-to-1 assignment", that is, each worker is assigned exactly one task and each task is assigned to exactly one worker. The *generalized* assignment problem (GAP) relaxes this requirement and allows multiple (or no) tasks respectively facilities to be assigned to one worker respectively location. It introduces capacities and demands that limit the assignment of one to the other. The GAP can be said to model $W$ workers that are assigned $T$ tasks. Worker $i$ is equipped with a capacity of $b_i$ of a certain resource (e.g. time, financial budget) and requires $r_{ik}$ units of that resource to complete task $k$. The assignment of task $k$ to worker $i$ also incurs a cost $c_{ik}$. Similar to the QAP the GAP is NP-hard. The problem can be described mathematically as [RS75]:

$$\min \sum_{i=1}^{W} \sum_{k=1}^{T} x_{ik} \cdot c_{ik} \tag{2.7}$$

$$\text{s.t.} \sum_{k=1}^{T} x_{ik} \cdot r_{ik} \leq b_i \quad \forall i \in \{1, ..., W\} \tag{2.8}$$

$$\sum_{i=1}^{W} x_{ik} = 1 \quad \forall k \in \{1, ..., T\} \tag{2.9}$$

$$x_{ik} \in \{0, 1\} \quad \forall i \in \{1, ..., W\} \wedge k \in \{1, ..., T\} \tag{2.10}$$

The objective in equation (2.7) is to reduce the cost of the assignment. Constraint (2.8) ensures that the sum of the allotted resources to each worker does not overbook the available capacity. Constraint (2.9) ensures that each task is assigned to exactly one worker.

### 2.1.4 Generalized Quadratic Assignment Problem

The generalized quadratic assignment problem (GQAP) is also NP-hard and represents "the quadratic extension" of the GAP. It combines the objective of the QAP (cf. equation (2.3)) with the constraints of the GAP (cf. equations (2.8) to (2.10)). A slight simplification in the proposed formulation is that in the GQAP it is assumed that each facility has the same space demand, regardless of the location. Thus $r_{ik}$ does not have matrix form, but is described only as a vector with elements $r_k$, although it would not be difficult to extend the formulation. Also a cost conversion factor $e$ is defined that balances the importance of the installation costs $c_{ik}$ and the costs resulting out of the strength-distance product. Mathematically, the GQAP can be described as the following integer programming problem [PHGZ10]:

$$\min \sum_{i=1}^{W} \sum_{k=1}^{T} \left( x_{ik} \cdot c_{ik} + e \cdot \sum_{r=1}^{W} \sum_{s=1}^{T} x_{ik} \cdot x_{rs} \cdot w_{ir} \cdot d_{ks} \right) \tag{2.11}$$

$$\text{s.t.} \sum_{k=1}^{T} x_{ik} \cdot r_k \leq b_i \quad \forall i \in \{1, ..., W\} \tag{2.12}$$

$$\sum_{i=1}^{W} x_{ik} = 1 \quad \forall k \in \{1, ..., T\} \tag{2.13}$$

$$x_{ik} \in \{0, 1\} \quad \forall i \in \{1, ..., W\} \wedge k \in \{1, ..., T\} \tag{2.14}$$

Since only one of the permutation constraints is present, i.e. equation (2.13) describes that each facility must be assigned exactly one location, the solution representation cannot be a permutation. The assignment is a, so called, "1:n assignment" which can be described using an integer vector. In this vector the index describes the facility and the value describes the location to which this facility is assigned to.

### 2.1.5 Benchmark Libraries

Benchmark data are important for the scientific community to compare algorithmic approaches directly with each other on the same problem instances. However, there are a number of concerns that benchmark libraries should take into account.

a Coverage: The set of benchmark instances should cover the space of possible problem instances adequately. The data should not only be

sampled uniformly and independently, but various sampling distributions and interdependencies should be included. This also includes the size of the instances. A library consisting only of small problem instances does not allow exploring the scaling behavior of algorithms.

b Imbalance: The set of benchmark instances should not consist mostly of instances of the same "type", but the various types should be represented rather equally.

For the QAP there have been several benchmark libraries proposed which will be introduced briefly. Each of those libraries consist of several instances often with different characteristics. This is important, because as we will detail and observe later on, different characteristics favor different algorithm instances.

1. **QAPLIB** [BKR97] - a collection of problem instances from a wide variety of sources. It includes randomly generated instances as well as real-world instances such as keyboard layout or hospital location planning. A wide variety of distributions for the flow and distance matrices are being used which include sparse and asymmetric instances.
2. **DreXX** and **TaiXXe** [DHTT05] - a collection of problem instances from small to large size that are harder to solve for most metaheuristics.
3. **Microarray QAP** [dR06] - a collection of real-world instances from microarray layout problems.
4. **TSPLIB** [Rei91] - these are instances for the traveling salesman problem (TSP), but which can be solved in a QAP formulation as described above. Many of the TSP instances are however much larger than an average QAP instance. A reason is the different runtime complexities for computing the fitness. The TSP objective can be computed in linear time $\mathcal{O}(n)$ whereas the QAP objective grows quadratically with the problem dimension ($\mathcal{O}(n^2)$). Thus QAP instances with a size of 100 are comparable to TSPs with a size of 10,000 complexity-wise.

The QAPLIB is certainly the most elaborate, but also most dated library. Many of the instances can be solved in reasonable time nowadays by one method or the other and thus harder to solve instances were introduced more recently [DHTT05]. The 137 instances constituting the QAPLIB libary are summarized in Figure 1 and Table 1.

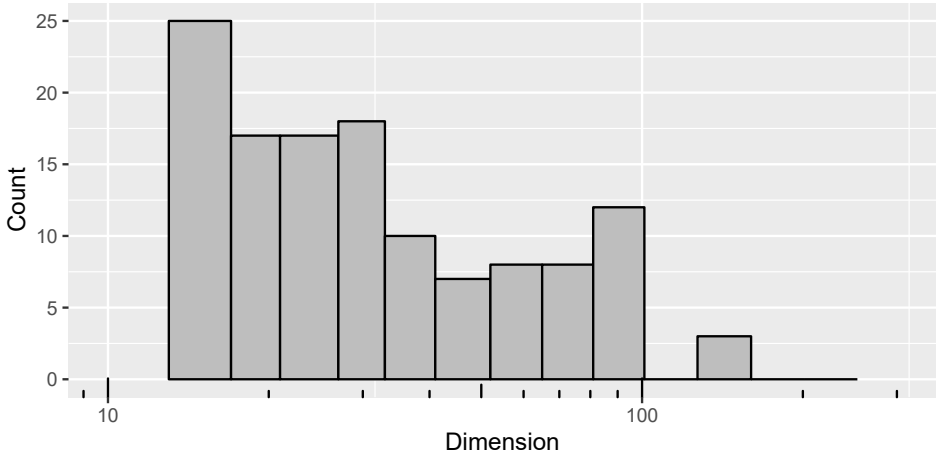



Figure 1: This histograms shows the frequency of the problem dimensions of all 137 instances in the QAPLIB. The bin sizes for the dimensions have been chosen in a logarithmic progression.

Table 1: QAPLIB constitutes of problem instances from a variety of sources and types. This table shows the number of instances per source and type, the average dimension of each group and its standard deviation.

| Row Labels | # Instances | ∅ Dimension | $\sigma$ Dimension |
|---|---|---|---|
| bur | 8 | 26.0 | 0.0 |
| chr | 14 | 17.6 | 4.2 |
| els | 1 | 19.0 | - |
| esc | 20 | 30.4 | 25.9 |
| had | 5 | 16.0 | 3.2 |
| kra | 3 | 30.7 | 1.2 |
| lipa-a | 8 | 55.0 | 24.5 |
| lipa-b | 8 | 55.0 | 24.5 |
| nug | 15 | 20.3 | 5.5 |
| rou | 3 | 15.7 | 4.0 |
| scr | 3 | 15.7 | 4.0 |
| sko | 13 | 81.1 | 22.0 |
| ste | 3 | 36.0 | 0.0 |
| tai-a | 13 | 38.0 | 27.7 |
| tai-b | 13 | 48.2 | 40.8 |
| tai-c | 2 | 160.0 | 135.8 |
| tho | 3 | 73.3 | 66.6 |
| wil | 2 | 75.0 | 35.4 |
| Total | 137 | 40.2 | 35.3 |

### 2.1.6 Literature Review

The QAP has been used in optimizing keyboard layouts, hospital facility layouts, microarray layouts, electronic wiring, and more [Ste61, Els77, HK01, dR06, Ben02]. The problem is also hard to solve even for small and moderate dimensions. The DreXX instances [DHTT05] are so hard that in a group of 10 different algorithm instances only one is able to achieve the optimum solution using up to 100,000,000 solution evaluation equivalents. Results will be shown and analyzed in more detail later in this thesis.

Research has focused on exact and approximate approaches. A common lower bound that is quick to compute for QAP problem instances is the bound by Gilmore and Lawler [Gil62, Law63]. Computing the bound involves sorting the weights and distance matrix in descending and ascending order respectively and then computing a linear cost matrix. This turns the problem into a linear assignment problem (LAP) that can be solved efficiently by the Hungarian algorithm [Kuh55] in $\mathcal{O}(n^3)$ time. The optimal objective value of this LAP is then a lower bound for the respective QAP.

Linearizations of the QAP are introduced by Kaufman and Broeckx (KB) [KB78], as well as Frieze and Yadegar [FY83]. Further advanced linearizations are described by Xia and Yuan [XY06] and Zhang et al. [ZBRM13].

Due to the difficulty of obtaining good solutions to the QAP using exact approaches many approximate algorithms have been described. Taillard described a tabu search algorithm. He noted that in some instances the search trajectory of tabu search becomes confined to a sub-optimal region of the search space. In his "robust tabu search" (RTS) algorithm [Tai91], Taillard describes an aspiration condition that would diversify the search trajectory after a number of iterations and thus escape the sub-optimal region. Diversification has been seen as a key aspect of solving QAP instances as efficient local search methods are available that may evaluate neighbors in constant time [Tai91]. Merz and Freisleben [MF00], as well as Drezner [Dre08] and Stützle [Stü06] describe memetic algorithms, respectively perturbation strategies to continue the search after a local optimum has been reached. The idea of memetic algorithms is to use a population of solutions as a diversifying element.

While the QAP has received much attention in the scientific literature, the generalized QAP (GQAP) is still fairly unexplored. Nevertheless, heuristic approaches to solve this problem have been described [CGLM06, MRS11].

## 2.2 Heuristic Optimization Algorithms

It is the nature of heuristic algorithms to choose among a set of alternative decisions. In the construction of a Hamiltonian cycle a heuristic could make decisions on the starting point as well as the next point in the cycle. In the combination of two different paths, cut points have to be chosen and duplicate or missing stops in the path have to be fixed. In the perturbation of a tour, a stop has to be removed and inserted between two other stops. Typically, such decision situations involve choosing one from a set of alternative actions. Heuristic algorithms then often include randomness in their decisions that alter an otherwise deterministic ranking. While this makes the algorithms more complex to analyze, stochastic heuristics can be reapplied in the hope to identify an improved solution. Stochastic heuristics are thus *more robust* in that they can achieve a better average performance if the best of $n$ runs is used as the final result.

The integration of heuristics into more complex algorithms has been described by the term *meta*heuristics [Tal09]. These algorithms describe a more general strategy in which individual heuristics are employed. For instance, the *canonical genetic algorithm* describes how a population of solutions is evolved by using *selection*, *crossover*, *mutation*, and *replacement* heuristics in an iterative manner. Each of these heuristics is described in form of required properties.

Many metaheuristic algorithms can be seen as *anytime algorithms* [Zil96]. They often iteratively improve solutions and can be interrupted at any time to return the best solution that has been found so far. In contrast to *contract algorithms*, that are given a fixed time, the *anytime behavior* of heuristic optimization algorithms is especially important [DLLIS11]. The computational budget given to a metaheuristic can be seen as an upper bound and the applicant of such a method expects to obtain good solutions well before this bound has been reached. Nevertheless, often in the scientific community these methods are compared as if they were contract algorithms looking at the final quality averaged over a number of runs after a fixed budget. This is adequate if the computational budget of the final application is known, however runtime requirements and limitations of the computation resources are not always clear from the beginning and might even change during the process of applying the algorithm in real-world situations.

Many metaheuristics are also characterized by offering a range of parameters that influence the behavior. Each parameterization essentially describes a different instance of an algorithm. The distinction between algorithms and instances in the light of parameters is important as only instances, and even more strictly only implementations of these instances, can be empirically compared against each other. These parameters serve a number of purposes which may be classified to

- limit the computational budget.
- scale the algorithm's performance to more difficult problems.
- define the strategy of a certain heuristic.
- balance the interplay of different heuristics.

An important characteristic that distinguishes approximate from exact approaches is the lack of a guarantee for achieving good solutions. Exact approaches, also referred to as complete algorithms [HS98, BR03], eventually compute the optimal solution given a finite amount of time and memory. Heuristics can only be said to *usually* work well, but without proof or guarantee and with a possibility of failing to work well. Each application needs to be studied empirically with respect to achievable solution quality and runtime. An a priori performance estimation of a certain heuristic applied to a certain problem is difficult to give. Progress has been made on evaluating the runtime performance of heuristic algorithms, but still a rigorous mathematical analysis is bound to simple problems and algorithms. An overview of various techniques and examples can be found in [LO16].

### 2.2.1 Real-world Applications

Heuristic and metaheuristic optimization algorithms have been used for some time successfully in finding good solutions to models that cannot be solved exactly in reasonable time or that cannot be described in closed mathematical form. A few problems such as vehicle routing problems (VRP) and its variants are prime examples where metaheuristics are very good performing algorithms. Vidal et al. describe a state-of-the-art memetic algorithm for solving VRPs [VCGP14]. They are able to deliver close to optimal solutions in the benchmark instances considered using reasonable amounts of computing power. Vonolfen et al. describe an island-based genetic algorithm variant that is able to achieve high quality solutions [VABW11]. Pisinger and Ropke describe an adaptive large neighborhood search (ALNS) that is applied successfully to VRPs [PR07].

A number of challenges arise when approaches are put in operational use, i.e. when the solution is actually transferred to the real world. Such an environment is, among others,

- *dynamic* - in a real-world environment there is continued change. New conditions may invalidate the optimality of a previous solution. For instance, an optimal schedule may be invalidated as resource availabilities or customer demand changes.
- *uncertain* - data we may have to describe the real-world is not exact. It may actually be described in terms of a probability distribution and there may be errors as any distribution we may have deducted is only an approximation to the true distribution that remains unknown.
- *asynchronous* - the real world and the model world are typically synchronized periodically. However, as has been stated, the real world changes. Anything the algorithm assumed about its state at the start of its calculation may not hold when it is finished. Especially, if the algorithm runs for a longer time.
- *stateful* - the real world does not always fulfill simple initial conditions, e.g. that all vehicles are at a depot, or that all queues are empty. Vehicles may be on their way to customers and the queue already contains jobs, some of which are being processed right now. Often it is difficult to acquire suitable data about this state, because either the rate at which the state changes is high and no automation exists or because there are simply no information systems that hold the system state in the required level of detail.
- *disorganized* - the execution of a solution in the real world requires coordination. An update to a tour must be communicated to the driver and errors/misunderstandings may arise during such a communication. This however can be seen as a special type of uncertainty as has been stated above.

Challenges such as those arising out of a dynamic and asynchronous environment may be mitigated somewhat by fast algorithms whether they be heuristics or exact algorithms. The less time these take to finish, the more likely initial conditions still hold when the algorithm terminates.

Another possibility to counter these situations could be to consider the dynamic and stochastic aspects of the real-world within models. This however comes at the expense of making them harder to solve. For instance, calculating

the expected tour length of the probabilistic TSP is of complexity $\mathcal{O}(n^2)$ [Jai85] as opposed to a complexity of $\mathcal{O}(n)$ when calculating the tour length of the deterministic and linear formulation. In addition, the parameters such as the probability of a customer appearing or not also have to be given.

In recent years the emerging topic of *open ended evolutionary algorithms*, of which ALPS [Hor06] may be a possible candidate, could allow to embed real-world disturbances as a form of uncontrollable mutation in the evolutionary process. Thus allowing open ended metaheuristics running in parallel to the real world and continuously exchanging state and solution information.

Further challenges in the application of heuristic algorithms arise in the selection problem that has been mentioned earlier. Some algorithms are better suited than others on different sets of problem instances. However, in the case of automated decision making or in embedded devices these algorithms run unobserved. For any designer or developer of that system it is quite difficult to decide a priori on the exact specification of the algorithm instance that is to be implemented. Typically, a few alternatives are ranked on a representative set of instances and then choosing e.g. the one with best average rank or according to some advanced voting method. Several competitions at conferences such as GECCO are organized this way. Better results can be achieved if a so called portfolio of algorithm instances is used.

### 2.2.2 Categorization of Metaheuristics

There are a variety of heuristic and metaheuristic methods that are created either with a metaphor or natural process in mind, or with the intention of achieving a certain goal, i.e. balancing exploration and exploitation. Several thoughts have been made on categorizing these methods so as to distinguish them and try to identify similarities and differences. Categories that are described in the literature are:

1. Nature-inspired vs. non-nature inspiration ([BPSV01, BR03, Tal09])
2. Population-based vs. single-point search ([BPSV01, BR03, Tal09])
3. Memory usage vs. memoryless methods ([BPSV01, BR03, Tal09])
4. One vs. various neighborhood structures ([BPSV01, BR03])
5. Dynamic vs. static objective function ([BPSV01, BR03])
6. Trajectory methods vs. discontinuous methods ([BPSV01])
7. Deterministic vs. stochastic methods ([Tal09])
8. Iterative vs. greedy methods ([Tal09])

Category 1 separates methods based on their source of inspiration. It is not expected that these categories are practically or theoretically relevant.

Category 2 on the other hand introduces a more interesting distinction. The introduction of a *population*, i.e. a collection of complete solutions that are fed into the variegation loop of such metaheuristics requires more memory, but enables a simultaneous exploration of a larger part of the search space. Solving certain problems may benefit from such a population.

Category 3 tries to distinguish between methods that memorize certain aspects of the search. The terminology used ("memory") however makes it slightly more difficult to categorize algorithms with respect to this group. All methods contain some sort of memory, the solution itself that is perturbed or crossed can be seen as a memory. Also memories come in different forms such as the previously mentioned population, but also in forms that are not related to solutions or their components such as a memory on the number of successful perturbations that have been applied with a certain perturbation strategy. Probably the most precise distinction in this class is based on whether the Markov property holds as [BR03] suggests. In [BPSV01] simulated annealing and greedy randomized adaptive search procedure (GRASP) are explicitly stated as memory-less stating that genetic algorithms and local search could be seen as memory-based algorithms and that this feature is "partially present". This is because several variants of genetic algorithms *can* be described as Markov chains [WZ99]. In [Tal09] no formal description is given, but again local search, GRASP and simulated annealing are listed as representatives of memory-less methods.

Category 4 introduces a distinction on the amounts of neighborhoods that are used. A neighborhood is a concept that relates solutions to each other. The neighborhood can be described as a function $\mathcal{N} : S \to \mathcal{P}(s)$ from the space of solutions $S$ to its powerset (assuming discrete solution spaces). Neighborhoods can also be seen as graphs where a change is the transition along a single edge in the graph. The use of just one such graph limits the ability of metaheuristics to explore the search space. This is mostly relevant for algorithms that iteratively improve solutions through small changes. For other strategies it may not be a relevant classification. The crossover heuristic in genetic algorithms is problematic to classify correctly here. Basically, it creates an extended graph that not only contains the solutions, but also all tuples of solutions. Also, construction-based metaheuristics such as GRASP or ACO can only be classified here with respect to the local search strategy that they

use. The construction part does not make use of the neighborhood concept, respectively it uses a concept of neighborhood on parts of a solution.

The distinction in Category 5 is not seen as a nature of the problem, but rather of the algorithm to modify the objective function. Only few methods currently exist that add their own bias to the objective value, e.g. guided local search (GLS) is mentioned by [BPSV01]. However, more recent approaches would include methods that make use of metamodeling techniques which approximate the fitness function with simpler mathematical models dynamically during the search [KL13]. In these approaches the objective value is not modified, but rather approximated with a change in the approximation at certain points in the search process.

In Category 6 the distinction is done between methods that iteratively perform only small changes to those that perform large changes, potentially combining parts of different solutions. While methods that can be classified as single-point search strategies (cf. Category 2) can fall into either class, population-based strategies are typically discontinuous methods [BPSV01].

The use of randomness in the search is the distinguishing criterion in Category 7. Deterministic algorithms would always return the same result given the same input, however almost all metaheuristics are stochastic algorithms.

Finally, Category 8 separates methods that construct solutions vs those that improve solutions iteratively. The term "greedy" as used by Talbi [Tal09] is probably not the best choice here as it is also used for strategies which deterministically choose the best among a certain ranking. In addition there are methods that are both constructing solutions and iteratively improving them. The aforementioned GRASP alternates between a constructive approach to build a complete solution and then an iterative method to improve it.

Apart from a conceptual presentation of different algorithm strategies, categorizing algorithms is an attempt to describe a relationship between algorithms. If two algorithms are put in the same categories potentially, there's some similarity to how they operate. It also carries the hope that some aspects of the observed performances may be present in one form, but not in the other. We may hope to find a certain range of problems which are, e.g. suited for population-based methods and where single-point methods converge to solutions of worse quality or vice-versa. As was already stated, some of these classifiers that have been mentioned are probably more likely to yield a significant distinction in the results than others. Overall however, it is questionable whether such categories are really useful in determining suitable algorithms. Metaheuristics are algorithms that carry the term "meta" in the name for the reason of being more widely applicable and for the reason of performing well in a range of problems. Also, they contain parameters that allows tuning the performance.

While comparing algorithms conceptually is an interesting task in itself, algorithms, or rather specific parameterized instances of algorithms are compared regarding their observed performance. The question then is which difference in performance is statistically significant (given a reasonable number of samples). Another form of significance can be attributed to the impact of the solution in the real-world. If cost savings or performance improvements in the real-world are rather low a performance difference is not relevant from the point of the real-world application. Often a major break-through in a field is only achieved when a major improvement is made and this means when it has a very high impact on the real-world.

Overall, the more interesting question beyond the comparison of algorithms is *why* a certain algorithm instance performs better than another and which aspects of the algorithms are responsible for the difference in performance. Is the population necessary, because diversification is needed, respectively, is a stronger perturbation necessary in trajectory-based methods? In doing that the hope is to learn about what strategies and ideas that these algorithms are composed of are responsible for the performance such that these can be composed of to form new algorithms that do not yet exist. We will further discuss this point and related methods in Section 2.3.

### 2.2.3 Further Classifications

In the aforementioned classes we can see that algorithms are characterized by certain variation concepts. A more complex classification system could be created considering their presence in the considered methods. In general, we can identify three different concepts:

- Solution assembly - better solutions can be described as the combination of the parts of other, primarily good, solutions
- Solution construction - better solutions can be created by iteratively picking the right solution component
- Solution perturbation - better solutions can be achieved by making changes to their components

It is visible that the solution itself is seen as an "arrangement" of smaller components. The identification of the best arrangement is the task of the heuristic or metaheuristic algorithm. In the assembly concept, it is further assumed that there exists a (hierarchical) structure among these components that can be discovered to identify cohesive parts and thus reduce the space of component arrangements.

In addition to variation, metaheuristics also define a selection and a memorization concept. Both concepts are used to compare solutions with each other. In general at some point an algorithm needs to make a decision which one of a number of solutions is the more favorable choice. The solution can then take part in a variation process or be memorized for use in a future variation step. The following concepts can be defined:

- Random - no preference or bias, all solutions are equal
- Probabilistic - solution properties have a stochastic influence; for instance, probability of selection depends on the quality of solutions
- Deterministic - solution properties have a deterministic influence; for instance, the best or most diverse solution is selected or memorized

These selection or memorization concepts can be applied on the quality of a solution, but also on other characteristics, such as the similarity to other solutions or the actuality of a solution, i.e. the iteration in which it has been created.

### 2.2.4 Variants of Metaheuristics

The variety of metaheuristic algorithms presently known is quite large. The introduction of new algorithms is still a goal of the research community, however often they are specialized algorithms tailored to a specific or a set of problems or to a specific solution encoding. In general, these algorithms can be classified into algorithms for single-objective problems and those that consider multiple objectives. While both classes share the same variation concepts, they differ in the details of the selection concepts that are used. In single-objective algorithms the identification of a "better" solution is achieved by a simple $<$ or $>$ comparison between two fitness values. In multi-objective algorithms concepts such as *Pareto dominance* in combination with *crowding* have been defined to create a ranking [DPA+02]. A further complication arises when multi-objective algorithms are compared with respect to their performance. Since the result of a multi-objective algorithm is a set of solutions, the comparison has to be done between sets. In this thesis, only single-objective problems are being treated, leaving the difficulties arising in multi-objective optimization for future researchers to solve.

In attempting to identify common concepts and common strategies among metaheuristics it can be observed that the variety in the details is quite large, although many common elements are present. The challenges that these algorithms attempt to solve can be summarized as the following questions.

- How to achieve a good balance between intensification and exploration?
- How to escape from local optima?
- How to maintain diversity in a population of solutions?
- How to adapt solution variation or sampling over the course of the search?

Simple algorithmic frameworks such as low-level local search ignore most of these questions. The goal of such algorithms is to find local optima as quickly as possible. Thus, they are often used as part of metaheuristics, as low-level local search is often a very effective method. The above mentioned questions arise when such a simple approach is not good enough and when algorithms are sought that perform better.

For instance, exploratory aspects have to be integrated into advanced algorithms in order to locate new and different solutions and thus also potentially better solutions. It is however not known in advance and also different from problem to problem whether such better solutions are located

**Algorithm 1** Generalized Evolutionary Algorithm Framework

```
1: procedure EA()
2:     pop ← Sample()                                  ▷ Initialize the population
3:     while not Terminate() do
4:         nextgen ← [ ]
5:         while Incomplete(nextgen) do
6:             parents ← Selection(pop)
7:             offspring ← Crossover(parents)
8:             Mutate(offspring)
9:             if Accept(parents, offspring) then
10:                nextgen ←+ offspring                ▷ Add offspring to next generation
11:            end if
12:        end while
13:        pop ← Replace(pop, nextgen)
14:    end while
15:    return pop
16: end procedure
```

rather close to other good solutions and thus are clustered in some region or whether they are scattered over the whole search space. The question on how to escape local optima is, essential for single-solution metaheuristics, while for population-based metaheuristics the question of diversity in the population is more relevant. It would be highly desirable that metaheuristics learn something about the problem they are solving and exploit that knowledge in the course of the search. In the following, several variants of metaheuristics are described that provide different answers to these questions.

### Evolutionary Algorithms

The class or framework of evolutionary algorithms (EA) comprises iterative improvement methods that usually maintain a population of solutions. A generalized framework for EAs is given in Algorithm 1. A set of functions, e.g., `Crossover`, `Mutate`, etc., need to be defined such that this algorithm may be applied to solve optimization problems. The description in Algorithm 1 is generalized and includes several variants such as offspring selection.

Genetic algorithms (GA) [Hol92, Mic99, AWWB09] are population-based and problem independent metaheuristics. They combine the assembly variation concept in form of a *crossover* heuristic with the perturbation concept in form of a *mutation* heuristic. The population is the variation pool from which solutions are *selected*. Selection is usually probabilistic with determin-

istic memorization. The GA gradually shifts the balance from exploration to intensification. As the population converges and become more and more similar a high sampling probability is spread over fewer solutions and thus this reduced space can be explored more exhaustively.

A theoretical background in form of the schema theorem [Hol92] exists for the so called canonical GA that describes that the defined length of schemas of above average fitness becomes larger over time. A schema is a mask that matches a subset of the solution space. This mask has wildcards (#) that may match all possible manifestations of a solution component and defined positions that match only a single manifestation. For instance the schema $H =$ `#, 1, 0, #, #` matches all binary strings that have a 1 at the $2^{nd}$ and a 0 at the $3^{rd}$ position. Each schema $H$ has a defined length $d(H)$ that is the difference between the last defined position and the first defined position. The fitness $f(H)$ is the average fitness of all solutions that match the schema.

GAs are nowadays applied to a large set of problems extending to different solution encodings. Both crossover and mutation can be described on a very general level. These heuristics have been defined for real and integer vectors, permutations, lists of lists, as well as for specific problems. A number of variations have been introduced in order to scale the performance of genetic algorithms. These performance improvements are dedicated to

- reduce the computation time.
- achieve better solutions.
- solver larger problems.

For instance, an island version or coarse-grained parallel variant was introduced to segregate the population into smaller sub-populations that frequently exchange solutions in a phase called "migration", but otherwise each island runs a variation loop of its own. This can be parallelized such that computation time can be reduced, but also achieves somewhat better solutions as diversity might be maintained longer [AT01, Alb05, ALN13].

Other variants of genetic algorithms such as offspring selection genetic algorithm (OSGA) [AW03, AWWB09] attempt to mitigate the problem of premature convergence. Premature convergence occurs when the algorithm loses diversity in its population and thus the relevant genetic information to assemble good solutions. Instead of comparing a new solution to the whole population it is only compared against the parent solutions. When an improvement in the combination of the genetic information of two parents, regardless of their

quality can be made, the solution is considered a worthy successor. OSGA then puts a requirement of allowing only worthy successors to enter the variation pool of the next generation. This creates a less competitive environment for offspring solutions and thus also diversity is reduced slower. On the other hand converge may be slowed as well. Similar concepts to offspring selection have been described as upward-mobility- and brood-selection [Alt94].

Evolution strategies (ES) [Rec73, BS02, HO01] are a family of problem independent metaheuristics. They are similar to local search, and in fact (1+1)-ES can be seen as a very strict specialization of local search, that in early formulations considered only the perturbation variation concept in form of *mutation*. ES specialize in adapting the strength of the perturbation depending on the state of the search. Also a population is introduced and later on, a *recombination* heuristic was added that describes an assembly concept. ES distinguish two types of recombination: (a) discrete and (b) intermediate. Discrete recombination is similar to the crossover heuristic, but is described more generally with an arbitrary number of parents. Intermediate recombination is a special case in which a centroid solution is computed among the parents. Selection may be deterministic (the best $n$) or random, while memorization is deterministic based on either the quality (called plus selection) or actuality of solutions (called comma selection).

While ES are considered problem independent, much of the research has been performed on real-valued problems and thus ES have not been as widely adopted as genetic algorithms. Nevertheless, the continued focus on real vector solutions has led to the development of advanced algorithms (or specific adaptation strategies). Covariance Matrix Adaptation ES (CMA-ES) is widely considered to be a state of the art solver for real-valued problems [HO01].

Scatter search (SS) [Glo99] is another evolutionary algorithm that is based on a combination of the assembly and perturbation variation concepts. It maintains two distinct populations that serve the explicit purpose of intensifying, respectively diversifying the search. Subsets are created between these two populations as well as between the solutions among a population. A new assembly variation called *path relinking* is introduced that samples not only a single solution out of the "assembly space", but explores a path between two solutions by iteratively adding components of the second solution to the first. Solutions will also undergo a number of strictly improving perturbations. Thus, in scatter search solution variation is not purely random, but greedy.

**Swarm Intelligence**

Cellular algorithms such as cellular genetic algorithm (cGA) [AD08] impose a certain structure on the population, but are otherwise similar to non-cellular evolutionary algorithms. Instead of allowing arbitrary pairings and mating pools to form, the cGA aligns individuals on a grid. Mating is then restricted to only those individuals in neighboring cells. Thus, the propagation of genetic information in the population has swarm-like behavior.

Other swarm algorithms such as particle swarm optimization (PSO) [KE95] typically operate in a space of solutions where the concept of direction exists. For instance, a point in a real-valued search space may be perturbed by moving it along a certain direction. In PSO the direction and the size of the change along the direction are two properties of a particle that are adapted during the run. PSO uses directional information towards the best point found so far and the best position of each particle. There are applications of PSO to combinatorial search spaces, e.g. random key encodings [Bea94].

Ant colony optimization (ACO) [DD99] is based on the collective behavior of ants. In more technical terms, an adaptive construction heuristic is described with a simple probabilistic adoption strategy. In each decision step an influencing weight is introduced that exerts a certain *bias* favoring decisions that have led to good solutions in previous iterations. In each iteration of the metaheuristic a collective of solutions is built by applying the construction heuristic. Then, the best solution is analyzed and the bias is slightly increased for all decisions that would reconstruct the observed solution and the process starts over. The method that exerts the bias can also be seen as a form of memory. In the case of ACO this is usually encoded in form of a matrix which represents the probability of choosing the options given in the columns having already chosen the option given in the row.

**Neighborhood Strategies and Memory-based Methods**

Variable neighborhood descent (VND) as well as variable neighborhood search (VNS) [MH97] are both algorithms that can been seen as a direct specialization of local search. These algorithms specialize in making use of multiple and different perturbation heuristics that are applied one after another repeatedly in order to find better local optima. In VNS the different perturbation heuristics are to diversify the search and explore new parts of the search space while in VND these are used for intensification.

**Algorithm 2** Iterated Local Search Framework, adapted from [LMS10]

```
1: procedure ILS()
2:     sol ← Sample()                          ▷ Initialize the solution
3:     history ← InitHistory(sol)              ▷ Initialize the search history
4:     Localsearch(sol, history)
5:     while not Terminate() do
6:         sol′ ← Perturb(sol, history)
7:         Localsearch(sol′, history)
8:         if Accept(sol′, history) then
9:             sol ← sol′
10:        end if
11:        UpdateHistory(history, sol)
12:    end while
13:    return sol
14: end procedure
```

VNS and VND are also very similar to iterated local search (ILS) [LMS10]. ILS can be viewed as a more general algorithmic description of which VNS would be a specialization. There is on the one hand the iterated local search algorithm that can be summarized as alternating between a perturbation phase and a local search phase. On the other hand, many algorithm variants such as VNS, VND, and ILS fit into a more general local search framework. A pseudo-code of that framework is shown in Algorithm 2. Any trajectory-based method can be used within the local search phase, typically a low-level local search as outlined in Algorithm 3 is used. Similar for perturbation, although exploiting multiple neighborhoods in a structured way would classify the algorithm as VNS. Finally, an acceptance criterion decides whether the next iteration is based on the just found solution.

Also introduced in the ILS framework is a memory component, i.e. in form of the *history*. This object is initialized, queried, and updated in various operators of the algorithm. Naturally, the use of memory creates more complex interactions and dependencies in the algorithm and between operators. For instance, some operator, e.g. `Perturb`, writes something into the history, while another, e.g. `Localsearch`, reads it.

Tabu search (TS) [Glo86] is such a "memory-driven" algorithm and makes extensive use of the search history. This metaheuristic extends the best-improvement local search framework in that it remembers previously changed solution components and allows reverting those changes only if a certain *aspiration condition* is met. Typically, these conditions state that a substantial

**Algorithm 3** Low-Level Trajectory-based Search Framework

```
1: procedure LLTSF()
2:     sol ← Sample()                          ▷ Initialize the solution
3:     history ← InitHistory(sol)              ▷ Initialize the search history
4:     while not Terminate() do
5:         N ← Neighborhood(sol)               ▷ Efficient partial evaluation of moves
6:         move ← Select(N, history)
7:         UpdateHistory(history, sol, move)
8:         Apply(sol, move)
9:     end while
10:    return sol
11: end procedure
```

improvement needs to be achieved. Often tabu search is implemented as a fully deterministic algorithm, only the initial solution, which could also be seen as input, may be randomly sampled. TS is often described to employ deterministic selection and exhaustive enumeration of the neighborhood. It describes several memories such as short-term, medium-term, and long-term to provide trade-off between exploitation and exploration of the search space. Basically, it can be fit into the local search framework shown in Algorithm 2. However, tabu search uses a much smaller part of the framework and due to the interactions with the search history, operators are more tightly integrated. Tabu search may be better fit into a low-level trajectory-based search framework as shown in Algorithm 3.

Such low-level trajectory-based search algorithms may be employed in form of `Localsearch` operators within the iterated local search framework. These algorithms are more low-level than other metaheuristics as the *move* object describes changes to solution components rather than full solutions and often an efficient delta calculation is required to calculate the fitness of the change in shorter time than that of the whole solution. Otherwise, the exhaustive enumeration of the neighborhood is a costly endeavor. In an implementation also the enumeration of `Neighborhood` and the selection of the next move (`Select`) are tightly integrated in that both are performed simultaneously. However, the generic overall definition does not put into question that tabu search is a metaheuristic.

## 2.3 Fitness Landscapes

Fitness landscapes are, among others, of primary interest in the analysis of algorithm performance. Properties such as the modality of a landscape, its ruggedness or deceptive properties for greedy search algorithms are to be measured and analyzed. The typical mental models that are used to describe fitness landscapes seek to invoke images of riffs, cliffs, valleys, peaks, and other characteristics. But while these properties may indeed be present in fitness landscapes, their detection is difficult and their relation to search performance is still to be explored in more depth. Formally, a fitness landscape is given by the triple $\langle S, N, f \rangle$ where $S$ is the space of solutions, $N$ is the neighborhood relation between the solutions in $S$ and $f$ is the function that assigns each solution a fitness value, i.e. the "height" in the landscape.

An example of a fitness landscape is shown in Figure 2. For higher dimensional problems or in non-metric search space such figures can hardly be shown in such a comprehensive state anymore. The function that generated that landscape is given in Equation (2.15).

$$f(x,y) = \cos^3\left(\frac{x * y}{\pi}\right) \cdot \sin^2(x) \cdot \sin(y) \cdot 2x \cdot (2\pi - x) \cdot (y - 2\pi) \quad x, y \in [0; 2\pi] \tag{2.15}$$

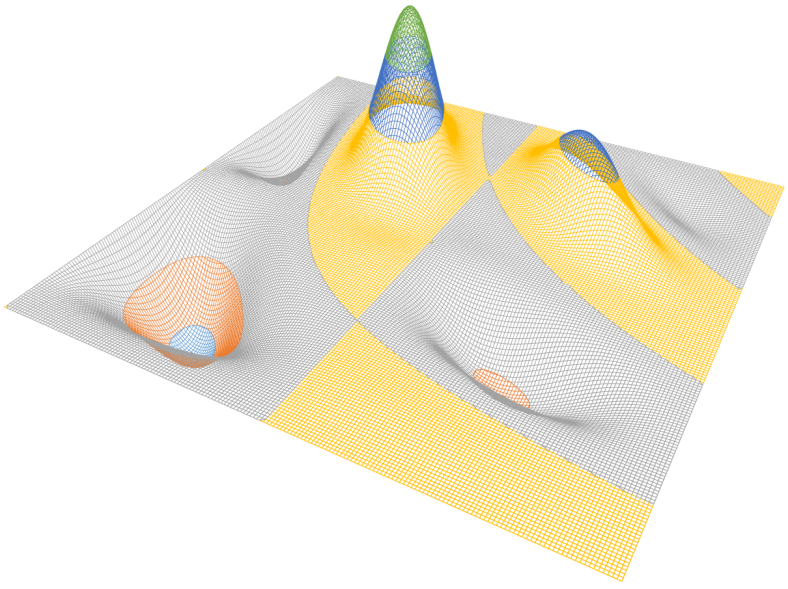


Figure 2: shows an exemplary fitness landscape of two continuous variables. The height of the landscape is the function's value that is to be maximized.

It may seem as if finding minima in this landscape is harder than identifying maxima due to the number of minima. In 1000 runs of a CMA-ES algorithm instance with initial $\sigma = \frac{2\pi}{5}$ and a population size of 10 the algorithm does converge in 50 generations to one of the four minima in nearly all of the runs. The minima are found in $\approx (60\%, 25\%, 5\%, 10\%)$ in order of best to worst minima of the cases using uniformly distributed random starting points. When maximizing the landscape with the same algorithm instance the global optimum is found in 69% of the runs. The $2^{nd}$ highest peak however is achieved in only 19% of the runs and in 11% of the runs the search converged into one of the flat area in the front of Figure 2. So number of local optima does have an effect, but it also depends on how bad these local optima are. In the case of maximization, the presence of the rather flat area is also a challenge. Ideally, one would say that a negative correlation between the quality of a local optimum and its basin of attraction contributes to search difficulty [PAB10]. Basins of attraction are however difficult to approximate in large search spaces with many local optima. A suitable model to describe the landscape in terms of optima and basins is given in the form of local optima networks (LON) [OVDT14]. In a LON a large basin may be apparent as a strong edge from a given node (=local optimum) to itself. Local optima networks are described in more detail later in this chapter.

Fitness landscape analysis has the goal of describing this and other landscapes with respect to several categories that are believed to have influence on algorithm performance. The community talks about categories such as ruggedness, modality, neutrality, deceptiveness, and others that are on the one hand difficult to define formally and on the other hand not independent to each other. These terms must be seen as high-level concepts that are linked with several concrete measurements.

### 2.3.1 Ruggedness

The literature is quite wordy when it comes to definitions on *ruggedness*. Here we review several defining statements that have been made by various authors that have done work in the field of fitness landscapes.

- "In essence, it can be described as the frequency of changes in slope from "up-hill" to "down-hill" or simply the number of local optima." [PA12]
- "A fitness landscape is said to be rugged if the landscape consists of many peaks, and if there is low correlation between neighboring points" [MF00]

- "[..] whether the selective surface or fitness landscape is smooth, containing a single global fitness optimum, or rugged, where selective constraints on differing mutational trajectories create multiple local fitness optima" [KS11]
- "In a rugged landscape, the fitnesses of neighbouring solutions are less correlated and thus, it is harder for a search method to infer a search direction from previous solution quality" [MGA17]
- "[..] It is said that a landscape is rugged if the number of local optima is high. The fitness distribution specified another property of landscape ruggedness that relates to the variety of landscape forms such as ridges, cliffs, peaks and others." [VFM03]
- "Generally, the degree of ruggedness can be estimated from the average of fitness correlations between parents and offspring or the fitness distance autocorrelation function obtained by using a random walk." [Kat14]
- "The number of local optima is a measure for the ruggedness of landscape." [Sta02]

From this sample of statements on the concept of ruggedness we can extract that there are actually two interesting properties. One is related to the change of fitness in the neighborhood while the other, rather clearly defined, is the number of local optima in the landscape. These properties are somewhat correlated, though one could have very high change in fitness and still have only single extreme point in the landscape and on the other hand have multiple optima, but a rather constant change in neighboring fitness.

But fitness landscape is not only concerned with a conceptual description and thus measures are defined that should relate to these concepts. Low and high values of these measures should represent a rugged or, its opposite, smooth landscape. Among the most well-known measures that relate to ruggedness are the autocorrelation coefficient ($\xi$) and the autocorrelation length ($l$) [CLA12]. The autocorrelation length conjecture [Sta02] states that a higher $\xi$ or $l$ describe a smoother landscape which in turn should correlate with a lower amount of local optima. However, in their analysis on the QAPLIB Chicano et al. observe that

"[..] the autocorrelation length conjecture can be applied only when the comparison is performed over instances with the same size $n$ and, in general, it is not true that the higher the value of $l$ the easier to solve the instance, since the largest instances are usually the most difficult ones and have the highest value for $l$ (and $\xi$). A good indicator of the difficulty of an instance could be

the pair $(n, l)$." [CLA12]

One of the reasons for this result is that autocorrelation does not consider the extent of a change. Every move is of unit size regardless of the dimension. However, the effect becomes smaller when swapping a constant number of items in a permutation that grows with problem dimension. If we are able to express the change's "strength" we can normalize these measures in a way that it becomes independent of the problem dimension.

In addition to autocorrelation the literature discusses a number of features coming from an information theoretic analysis of the quality trail resulting from a certain walk in the landscape. The features that have been introduced by Vassilev et. al [VFM00] are

- Information Content
- Partial Information Content
- Information Stability
- Density Basin Information

More formally as Vassilev states [VFM00], these features are defined over a sequence of fitness values $T = \{f_t\}_{t=0}^{n}$ as obtained from a walk on the landscape. This sequence can be transformed into a string $S(\varepsilon) = s_1 s_2 s_3 ... s_n$ where $s_i \in \{\searrow, \rightarrow, \nearrow\}$. These characters represent a downhill move ($\searrow$), a neutral move with respect to $\varepsilon$ ($\rightarrow$), and an uphill move ($\nearrow$) respectively. The symbols can be combined to 9 different shapes or symbols by considering two subsequent characters $s_i s_{i+1}$ as given in Table 2.

Table 2: The combination of characters from the set $\{\searrow, \rightarrow, \nearrow\}$ to symbols.

| | $\searrow$ | $\rightarrow$ | $\nearrow$ |
|---|---|---|---|
| $\searrow$ | $\searrow\searrow$ | $\searrow\rightarrow$ | $\searrow\nearrow$ |
| $\rightarrow$ | $\rightarrow\searrow$ | $\rightarrow\rightarrow$ | $\rightarrow\nearrow$ |
| $\nearrow$ | $\nearrow\searrow$ | $\nearrow\rightarrow$ | $\nearrow\nearrow$ |

Vassilev further introduces the frequency of such a symbol in the resulting string as $P_s = \frac{n_{[s]}}{n}$, i.e. the relative frequency of symbol $s$ appearing among all symbols $n$ in string $S$ [VFM00]. The symbols are then grouped into two mutually exclusive sets. The first set, denoted $A$ in Equation (2.16), contains all symbols where both characters are not alike. In contrast, the second set, denoted $B$ in Equation (2.17), contains the remaining symbols, i.e. those where both are alike.

$$A = \{\searrow\rightarrow, \searrow\nearrow, \rightarrow\searrow, \rightarrow\nearrow, \nearrow\searrow, \nearrow\rightarrow\} \tag{2.16}$$
$$B = \{\searrow\searrow, \rightarrow\rightarrow, \nearrow\nearrow\} \tag{2.17}$$

The information content is then computed as the entropy with respect to symbols of set $A$ (cf. Equation (2.18)) while density basin information is computed with respect to symbols of set $B$ (cf. Equation (2.19)).

Partial information content is slightly different to the other two. It is obtained by constructing a reduced sequence $S'(\varepsilon) = s_1 s_2 ... s_\mu$ with only those characters $s_i \neq 0 \wedge s_i \neq s_{i+1}$. The reduced string $S'(\varepsilon)$ thus consists only of $\searrow\nearrow$ and $\nearrow\searrow$ symbols and has a length of $\mu$. Partial information content is then the ratio of the lengths of $S'$ and $S$ (Equation (2.20)) [VFM00]. Finally, information stability is the highest epsilon for which the landscape becomes completely flat, i.e. the largest difference between any consecutive values in $T$ (Equation 2.21) [VFM00].

$$\text{InformationContent: } H(\varepsilon) = \sum_{a \in A} (P_a \cdot \log_6(P_a)) \tag{2.18}$$
$$\text{DensityBasinInformation: } H(\varepsilon) = \sum_{b \in B} (P_b \cdot \log_3(P_b)) \tag{2.19}$$
$$\text{PartialInformationContent: } M(\varepsilon) = \frac{\mu}{n} \tag{2.20}$$
$$\text{InformationStability: } \max(\text{abs}(f_t - f_{t+1})) \quad \forall t \in [0; n-1] \tag{2.21}$$

In the further course of this thesis, the feature *ic* describes the value obtained by Equation (2.18), *dbi* describes the value obtained by Equation (2.19), *pic* describes the value obtained by Equation (2.20) all with $\varepsilon = 0$, while *is* describes the value obtained by Equation (2.21).

**Information Analysis Example**

An example shall be given in the following. Consider the sequence of fitness values $(1, 2, 1, 3, 2, 4, 2, 1, 1, 2, 4, 4, 3, 1)$ which is translated into the following sequence of characters ($\nearrow\searrow\nearrow\searrow\nearrow\searrow\searrow\rightarrow\nearrow\nearrow\rightarrow\searrow\searrow$). We can then compute the frequency of the symbols as given in Table 3. Then the information characteristic analysis results in the values given in Equations (2.22)-(2.25).

Table 3: Resulting symbol frequencies from the example in the text.

| | Symbol | Frequency | $P_s$ |
|---|---|---|---|
| $A$ | $\searrow\rightarrow$ | 1 | $0.08\overline{3}$ |
| | $\searrow\nearrow$ | 2 | $0.1\overline{6}$ |
| | $\rightarrow\searrow$ | 1 | $0.08\overline{3}$ |
| | $\rightarrow\nearrow$ | 1 | $0.08\overline{3}$ |
| | $\nearrow\searrow$ | 3 | 0.25 |
| | $\nearrow\rightarrow$ | 1 | $0.08\overline{3}$ |
| $B$ | $\searrow\searrow$ | 2 | $0.1\overline{6}$ |
| | $\rightarrow\rightarrow$ | 0 | 0 |
| | $\nearrow\nearrow$ | 1 | $0.08\overline{3}$ |

$$\text{ic} : -\frac{1}{12} \cdot \left(4 \cdot \log_6\left(\frac{1}{12}\right) + 2 \cdot \log_6\left(\frac{2}{12}\right) + 3 \cdot \log_6\left(\frac{3}{12}\right)\right) \approx 0.8224 \quad (2.22)$$

$$\text{dbi} : -\frac{1}{12} \cdot \left(\log_3\left(\frac{1}{12}\right) + 2 \cdot \log_3\left(\frac{2}{12}\right)\right) \approx 0.4603 \quad (2.23)$$

$$\text{pic} : \frac{8}{13} \approx 0.6154 \quad (2.24)$$

$$\text{is} : 2 \quad (2.25)$$

In summary, the main idea of information analysis features is to compute the entropy of various shapes in overlapping subsections of the quality trail. An uphill move followed by a downhill move would be seen as a sign of ruggedness, whereas two succeeding uphill moves would be seen as a sign of smoothness. Two consecutive moves that do not appear to change the solution quality would be seen as an indication of neutrality. The parameter $\varepsilon$ controls the degree to which changes will appear neutral and thus gradually flattens the landscape if increased. For each of the features mentioned above there exists an $\varepsilon$ such that the feature becomes maximal. This is then called the "peak" of the feature and may also be used to characterise the landscape. Figure 3 shows these features as functions of different $\varepsilon$-levels.

### 2.3.2 Modality and Funnels

It has been mentioned previously that a multi-modal landscape is perceived to be also rugged. In that sense multi-modality was synonym for the presence of multiple local optima. However, local optima are defined always with respect to a certain neighborhood. In the domain of continuous function optimization

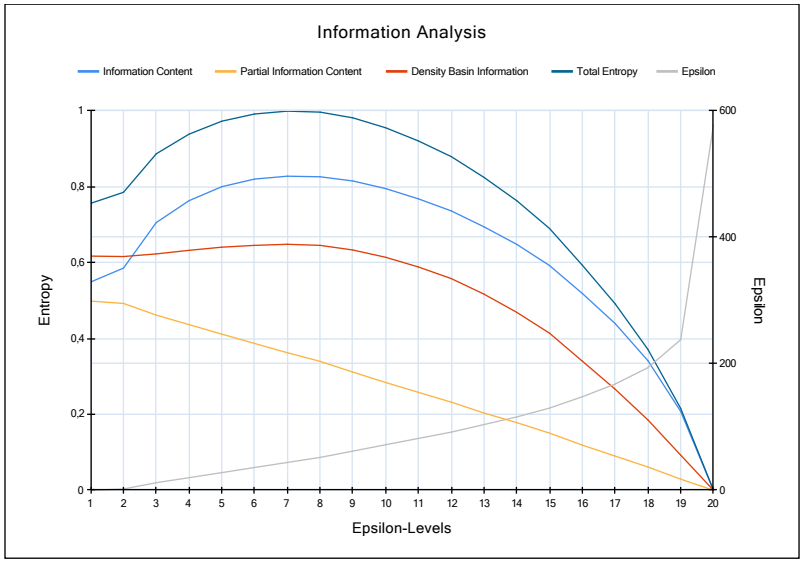
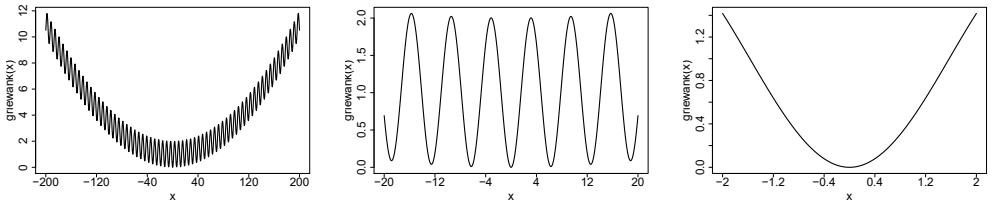



Figure 3: This chart shows the features information content, partial information content, and density basin information depending on the $\varepsilon$ level. Information stability is the $\varepsilon$ right before entropy becomes 0, i.e., the landscape would appear flat. All features, except partial information content reach a peak when the entropy is maximized. [Pit13]

we can best observe that functions generate different landscapes at different resolutions. Figure 4 presents three different views of the one dimensional Griewank function [Gri81]. The resolution is increased each time by an order of magnitude. There are many local optima in the Griewank landscape in Figure 4a and thus it would be regarded as highly multi modal and also rugged, however, all of these optima are part of one bigger basin shape. To describe such an aspect of "modality" in fitness landscapes the term "funnel" was coined. According to Ochoa and Veerapen [OV17] the origin lies within the protein folding community. They cite Doye et al. as

> "A key concept that has arisen within the protein folding community is that of a funnel consisting of a set of downhill pathways that converge on a single low-energy minimum." [DMW99]

Malan and Engelbrecht give a very compact description "a funnel in a landscape is a global basin shape that consists of clustered local optima" [ME14b] which is in turn derived from Sutton et al. [SWLH06]. Kerschke et al. also note that "it is not exactly defined what constitutes a funnel structure." [KPWT15] but further note that "the distribution of optima and the correlation of their




(a) Coarse view (b) Medium view (c) Fine view

Figure 4: Different views of the one dimensional Griewank test function with different levels of detail. A global shape is visible when viewed on a coarse level of details, at an intermediate level of detail it is difficult to determine a global optimal solution and on a very fine level the function is smooth and local optima should be attained quickly.

objective value" influence the notion of a funnel. In the case of Griewank we may not notice a clustering of local optima as they're uniformly distributed, but we may observe a global basin shape and thus a correlation between quality of local optima and distance to global optimum. In the landscape generated by a moderate view of the landscape (cf. Figure 4b) however the global basin disappears and we would conclude that there are multiple funnels.

Another example is given by a visualization of the solution space of the two dimensional Rastrigin function as shown in Figure 5. In two dimensions the numerous local optima and the high ruggedness of the landscape are easily visible. The landscape is thus considered to be highly rugged. However, it is also apparent that there exists a global structure in form of a paraboloid, that the optimum lies in the center of that paraboloid, and that there does not exist a second such structure with a barrier in between them [ME14b].

A measure that describes the extent of modality and the presence of multiple funnels is called the dispersion metric (DM) [LW06]. The idea is to capture an unbiased sample of the solution space ($S$) and relate the pair-wise distances among all solutions in the sample to the pair-wise distances between only the best solutions ($S^*$). Lunacek and Whitley described this in form of a plot that shows dispersion among the best, e.g. 10% over an increasing number of sampled solutions [LW06], while Malan and Engelbrecht describe a single measure of difference [ME14b]:

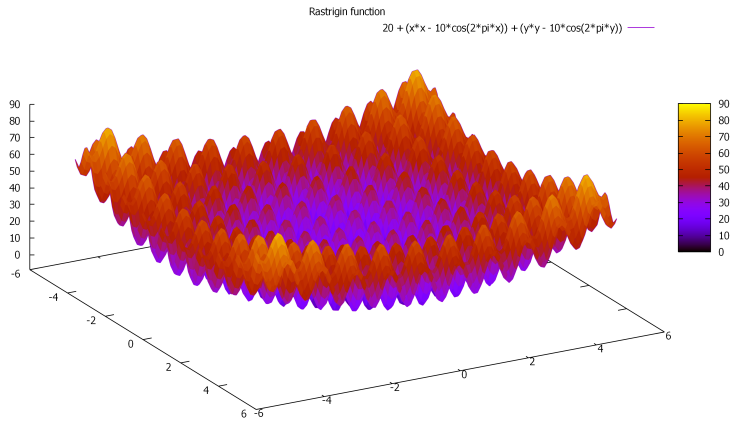



Figure 5: Visualization of the solution space of the two dimensional Rastrigin test function.

$$\texttt{DM:} \quad S \to \texttt{disp}(S^*) - \texttt{disp}(S) \qquad \text{where } S^* \subset S \tag{2.26}$$

$$\texttt{disp:} \quad S \to \frac{1}{n \cdot (n-1)} \cdot \sum_{s \in S} \sum_{k \in S, s \neq k} d(s,k) \qquad \text{where } n = |S| \tag{2.27}$$

The idea of the difference is that negative values indicate that the average pair-wise distance of the best solutions is smaller than that between all solutions in $S$ and thus that the better ones are clustered. Positive values on the other hand indicate the presence of multiple funnels as the average pair-wise distance between the best solutions is larger and thus fewer solutions are spread out over a larger range.

Further methods for analyzing funnels such as measures derived from nearest-better clustering heuristics are described by Kerschke et al. [KPWT15]. They propose five different measures based on a nearest neighbor distance. Similarly to the dispersion metric the features are computed comparing two sets, one that considers the whole population and another that considers distances only to better neighbors.

### 2.3.3 Deceptiveness

This concept is not necessarily a property of the landscape alone, but includes the exploration strategy of a certain algorithm. A landscape is deceptive if better solutions are found at an increasing distance to the global optimum or

to solutions of a certain desired quality. For a sufficiently greedy strategy it is thus highly unlikely to locate the global optimum and it will be trapped in an inferior part of the solution space.

One measure introduced in the literature to describe the extent of deceptiveness of a certain landscape is called the *fitness-distance correlation* or FDC.

The FDC describes the correlation between the fitness of solutions and its distance to a certain desired solution, e.g. the global optimum. A highly positive correlation coefficient indicates that better solutions are also closer to the desired solution. Thus, a simple greedy strategy should be sufficient to eventually sample the desired solution. On the other hand a negative correlation coefficient indicates that the desired solution may not be reached as the chance to sample it decreases with increasing distance.

### 2.3.4 Isotropy

Isotropy is often an assumption in the study of fitness landscapes. Currently, only little research has been attempted in describing isotropy. Anisotropic landscapes would be for example rugged in a certain part of the solution space and appear smooth in another part.

A simple example of anisotropic landscapes that appear in practice may be seen in the form of constrained optimization problems. Such a problem parts the solution space into a set of feasible solutions and a set of infeasible solutions. Depending on the type of penalization used the infeasible solution space may appear smooth in comparison to the feasible space and overall there may be a steep cliff between these two regions.

### 2.3.5 Separability

Finally, not necessarily a property of the landscape, but nonetheless important in the context of studying algorithm performance on landscapes is the concept of separability. A fitness landscape may be considered separable if a solution can be found by considering to optimize each dimension regardless of the others. The aforementioned Rastrigin function is separable. For each dimension the global optimum is always at 0 regardless of the values of the other dimensions. A separable function thus may also be considered as highly isotropic.

## 2.4 Landscape Analysis

In general we distinguish between exploratory and exact landscape analysis. Although the literature on exploratory analysis is concerned mostly with the study of real-valued solution spaces, in this thesis I think of it as a more general term that applies to all solution spaces. Exploratory (fitness) landscape analysis (ELA) relies on drawing samples from the fitness landscape and analyzing those samples. Thus, research is concerned on the one hand with sampling strategies and on the other hand with analysis methods that calculate features out of those samples. The term ELA has been coined by Mersmann et al. and also linked with specific features [MBT+11]. Some of those features are natural to apply to problems in the continuous domain and are not as easily transferred to combinatorial search spaces in general.

In exploratory analysis, the fitness function is viewed as a black box. The solution encoding is known, and a certain neighborhood relation is assumed, but the landscape heights, gradients, etc. have to uncovered. For this purpose several sampling techniques are described [MBT+11, PA12], including but not limited to random and adaptive walks, up/down walks, or neutral walks. The results of such walks are paths through the fitness landscape which are subject to further analysis. Sometimes only a collection of samples without the notion of connectedness in a path are analyzed and instead relying on some sort of distance.

In exact analysis, the fitness function is a white box. It is mathematically proven that for a certain type of function and/or neighborhood relations certain properties exist. For instance, it has been shown to compute the average quality $\hat{f}(s)$ of all neighboring solutions $s \in N(x)$ to a solution $x$ for all problems that may be decomposed into, so called, elementary landscapes [WSH08, CWA11]. In another result, it has been shown that the autocorrelation coefficient and the correlation length can be computed exactly in $O(n^8)$ for any instance of the QAP using a "swap2" neighborhood [CLA12].

While exact analysis are helpful and the results show potential to strongly influence new and improved methods, their application is often limited. Sometimes findings can be applied to single problem definitions only, sometimes they are more broader applicable. In this context, the elementary landscape decomposition, as mentioned above, shows great potential in that it seeks to formalize theories that hold for a larger class of optimization problems. In this thesis the focus is on exploratory analysis as it is more generally applicable.

In exploratory landscape analysis, characteristics of the landscape are estimated with the use of samples. Three different approaches can be distinguished in the way the samples are retrieved. On the one hand, a set of solutions are sampled from the solution space based on a random element, e.g. using a latin-hypercube sampling strategy [MBT+11]. I call this approach *bag analysis*, because essentially, the result is a "bag of solutions" and there is no a priori known relation between these solutions. On the other hand, trajectories or paths of solutions are created by sampling a starting solution and then progressively sampling from the neighborhoods. A number of these, so called, "walks" have been described in the literature [PA12]. I call this approach *path analysis* in this thesis as the sequence of the sampled solutions is important and this characteristic of the sampling process is exploited in the features that are calculated. Finally, *network analysis* considers not only a single path, but all pairwise interactions and thus approximates the landscape by a graph. In the following these approaches are detailed.

### 2.4.1 Bag Analysis

The most well-known example of ELA using a bag analysis approach is described by Mersmann et al. [MBT+11]. Here I just briefly summarize the approach. The bag of solutions that is obtained by latin-hypercube sampling is termed $D^s$ and consists of a vector of solutions and a corresponding vector of fitness values. It is argued that the size of the sampled bag should depend on the dimensionality of the problem to "account for the increasing problem complexity" [MBT+11]. There are 50 features which fall into six categories:

- Convexity - to which degree a linear combination of the input vector agrees with a linear combination of the fitness
- Fitness-distribution - skewness and kurtosis of the fitness values as well as an estimation of the number of peaks
- Levelset - quality of various classifiers from discriminating bad solutions from good ones using a certain threshold
- Meta-Model - quality of linear and quadratic regression models for predicting the fitness values
- Local search - measures basin size and local optima clusters based on Nelder-Mead local search
- Curvature - features based on the estimated gradients

Some of these features are more complex to compute than others. For

instance, fitness-distribution can be calculated rather easily, but local search features are more expensive. It has been shown that, in the continuous domain, this approach results in features suitable for algorithm selection [BMTP12].

### 2.4.2 Path Analysis

A number of different walks have been introduced in path analysis that should briefly be reviewed here:

**Random Walk** : The random walk is a sampling strategy in which the next sample is a randomly chosen neighbor of the current sample. It is very simple, but requires a larger number of samples and often leads to poor characterizations [ME14a, BWA18]. A variant called *progressive random walk* [ME14a] was introduced for continuous problems, in which a certain "sampling direction" is enforced. The trajectory forms a wiggly path that bounces off at the domain limits. Progressive walks cover a bigger part of the search space without increasing step size too much.

**Adaptive Walk** : In the adaptive walk sampling strategy the next sample is chosen to be the best neighbor out of the neighborhood's randomly chosen subset. Adaptive walks are computationally more intensive as multiple neighbors need to be evaluated which are then also often discarded in the analysis.

**Up/Down Walk** : These walks are similar to adaptive walks, but whenever the current sample cannot be improved in quality, it reverses the search direction and instead focuses on deteriorating the sample's quality until no further deterioration is possible in which case the process is reversed again [PA12].

**Neutral Walk** : These walks attempt to explore the neutrality of a certain landscape. As introduced by [RS01] the walk starts with a randomly sampled configuration. It progresses by sampling from the set of neighbors that have the same fitness and which also increase distance to the starting solution over the distance from the current solution.

As [PA12] summarizes nicely, there are two requirements for these kinds of walks: (1) A neighborhood function $\mathcal{N}$, and (2) a selection strategy $\zeta$ for choosing a successor. The result is a sequence $\{x_i\}_{i=1}^n$ of solutions with $x_{i+1} = \zeta(\mathcal{N}(x_i))$ and we analyze the resulting sequence of fitness values $\{f_i\}_{i=1}^n = \{f(x_i)\}_{i=1}^n$.

Both bag and path analysis provide some low level characterizations which in turn may explain higher level characteristics which are relevant to optimization algorithms. The downside of bag analysis is, that space-filling designs such as latin-hypercube sampling are difficult to realize in general for combinatorial search spaces. The downside of path analysis is that each of these walks needs to be computed independently and thus a lof of samples need to be drawn. Also, adaptive, up/down, and neutral walks use subsets of the neighborhood which are more expensive to analyze. In addition, the subset size is a parameter of the walk which in turn increases complexity as the number of neighboring solutions in the subset has to be determined through experimentation. A further disadvantage, as has been noted already is that the evaluated neighbors are discarded from the analysis. For a difficult and long-to-evaluate simulation-based optimization problem, the number of solutions that we may sample and evaluate is highly limited and every solution is valuable. Also some walks, e.g. that make use of direction, can be used only in search spaces where such a concept is available. In Section 3 a new walk is introduced that features attributes from at least three of the above walks.

### 2.4.3 Network Analysis

A recent method of fitness landscape analysis is to generate so called local optima networks (LON) [OVDT14]. A local optima network is a graph of the solution space that includes solutions as their nodes and edges describe transition probabilities between those solutions. Naturally, if the whole solution space is considered, the graph would contain an extremely large number of nodes as often solution spaces grow exponentially with the problem dimension. Thus, the graph considers only local optima. This is a reasonable limitation as any solution that is not a local optima is undesired in that it can be improved through a small change. The same cannot be said about local optima for which there is no improving step within the neighborhood and a larger change need to be made, before a better solution could be achieved.

Generating a local optima network is usually performed by sampling local optima through repeatedly applying local search algorithms. This is of course a computationally very challenging effort. There are two types of edges described [OVDT14]: (1) basin-transition edges and (2) escape edges (introduced in [VDOT12]). The edge weight $e_{ij}$ in both describes the probability that a random walk moves between the local optima's basins of attraction. Basin-

transition edges describe this probability as the average *connectedness* between solutions of basin $i$ to solutions of basin $j$. Typically, two solutions are *connected* with strength $\frac{1}{|\mathcal{N}|}$ if one solution is within the neighborhood of another, where $|\mathcal{N}|$ denotes the size of the neighborhood. Escape edges describe this probability in terms of the ratio of solutions with distance $\leq D$ to basin $i$ are part of basin $j$. Escape edges are somewhat faster to compute as they do not require to enumerate the basins of attraction. The details on how to compute these edges are given in the respective references. An efficient implementation is described by [Fie18].

A number of measures from the domain of graph analysis are then computed from such a LON such as eigenvector centrality or clustering coefficients. These measures describe the structure of the graph and thus the structure of the local optima given a certain neighborhood [DVOT14].

### 2.4.4 Summary and Outlook

In this chapter, the fitness landscape was formally introduced, the motivation for its analysis was described and the methods were examined in more detail. Several possible short comings within the state of the art were mentioned that should be improved. Among others we would like to have a unified sampling strategy in order to minimize the computational overhead of different walks.

## 2.5 Performance Analysis

Metaheuristics are often compared within two domains. In a *fixed-budget* analysis we can limit their budget and measure the quality achieved when this budget is used up, while in a *fixed-target* analysis we can define a target quality and measure the required budget to achieve or surpass that target. Often the former is chosen, however it is suggested, the latter leads to better interpretable results. Among others, the performance with respect to a required budget, i.e., runtime, is "quantitatively interpretable", e.g. "algorithm A is two times better than B if it is able to achieve the same target in half the time" [HAM+16]. In addition, the domain of runtime is rather stable and its influences are known, while the domain of target or fitness values may change from one problem instance to another. For example, fitness values could be expressed in units of time in one problem instance and units of distance in another. Runtime, expressed, e.g., as elapsed wall clock time is influenced by implementation aspects such as programming language and parallelization, and hardware aspects. In order to have a more robust estimator of runtime, it is suggested to use calls to the fitness function as approximation of the runtime performance [HAM+16].

Thus, newer works compare the *expected runtime* (ERT) among various algorithm instances competing against each other and on various targets. This gives a good overview of how fast a method may be able to reach a certain goal. Along with the ERT is the empirical cumulative distribution function (ECDF) that is used to plot the results against each other. This provides a graphical way to compare performance. Additionally, there are box-plots for fixed-budget comparisons.

As has been described in section 1.1 comparison of heuristic and metaheuristic algorithms concerns several dimensions [Ric76, BB06]:

1. *efficiency* - concerns the runtime of an algorithm
2. *reliability* - concerns the quality of the achieved solution
3. *robustness* - concerns an algorithm's ability to achieve good quality among a range of problems
4. *simplicity* - concerns the complexity of an algorithm's implementation

Efficiency, reliability, and robustness are certainly among the most interesting and relevant when comparing heuristic optimization algorithms. Simplicity is a property that is often neglected, because the implementation is already

available prior to being able to test it. When comparing heuristic optimization algorithms, we would seek an algorithm that delivers the best value, in terms of solution quality, given the least computational effort over the most instances of a given optimization problem. Alas, the no-free-lunch (NFL) theorem states that we should not expect to find one algorithm that excels in all of these dimensions. But that we will rather find certain algorithms outperforming each other in different ranges of these dimensions. For instance, algorithm $A$ outperforms algorithm $B$ on a set of problem instances $P_a$, but the opposite is observed for the set of problem instances $P_b$. Or, even more interleaved, algorithm $A$ may outperform algorithm $B$ only during the first 5 seconds, after which algorithm $B$ finds solutions with better quality.

To conduct a well made comparison is quite often a complex task. First, one has to obtain an executable implementation of all methods that are compared against, otherwise relying only on published results, most likely executed under different conditions. Then, the (hyper-)parameters of the selected algorithms have to be chosen. This often constitutes a computationally intensive precursory study in which some of the problem instances are chosen for "hyperparameter optimization". Then, the selected algorithms are compared among a set of test problem instances. The significance of such a study is increased when the experiment

- makes use of a diverse range of benchmark instances. [SMvH11]
- gives each method the same computational time, making use of restart strategies for methods that are not able to utilize a large amount of computational time in one run. [HAM+16]
- includes a wide range of published algorithms.
- includes an explanation on the choice of algorithm instances, i.e. parameterizations of algorithms.
- includes a high number of repetitions to account for stochastic effects.

It is rather simple to compare algorithms with respect to runtime when a target quality has been defined. All algorithms can be run on the same machine and a stop watch is used to detect when the target quality has been reached. We say "rather simple", because if parallelism is introduced then the machine introduces a larger bias with respect to the degree of parallelism that it supports. In addition, if targets are not defined a priori, but the analysis should be done a posteriori with a range of targets, then all algorithms have to keep a logbook of which quality was achieved at which time.

Similar challenges arise when algorithms are compared with respect to achievable quality given a certain computational budget. Again, the simple case would be to run algorithms until the budget limit and then compare the quality. However, if the budget limit for an application may change (e.g. because computational power grows) again the aforementioned logbook has to be kept in order to perform analysis a posteriori.

In both cases a high number of repetitions have to be performed in order to have a reasonable estimation on an average over the runtime or the quality. Naturally, a high number of repetitions requires high amounts of computational power and if the gap is very small, a clear winner may not be identified. Thus often statistical tests are performed in order to determine if one is significantly better than the other. These tests have their own problems with respect to a misuse and overestimation of *p-values* [SF12, HHL+15].

But to describe the complexity of the algorithm is the fuzziest of all dimensions. There exists complexity measures of algorithms, such as lines-of-code (LOC), McCabe [McC76], etc. that attempt to describe one view on algorithm complexity. But algorithm complexity is by itself a multi-dimensional measure and concrete measures capture a few dimensions only.

### 2.5.1 Quality Distribution

The calculation of descriptive statistics on the achieved quality after a defined computational budget such as best, worst, and average quality together with the standard deviation are often listed in research papers. Sometimes it is the only type of analysis of the obtained results and given in tabular form or in box plots.

Analyzing and comparing final quality distributions is the most common way of evaluating metaheuristic performance. However it suffers from several drawbacks:

- Quality distributions show only a snapshot of the performance after a fixed computational budget. Properties of the search behavior such as convergence speed are neglected.
- Fair comparisons should be done using the same amount of computational budget for each algorithm instance. This is not always easy to achieve and may require retesting when only the final results have been recorded.
- The researcher defers the definition of meaningful target values to those

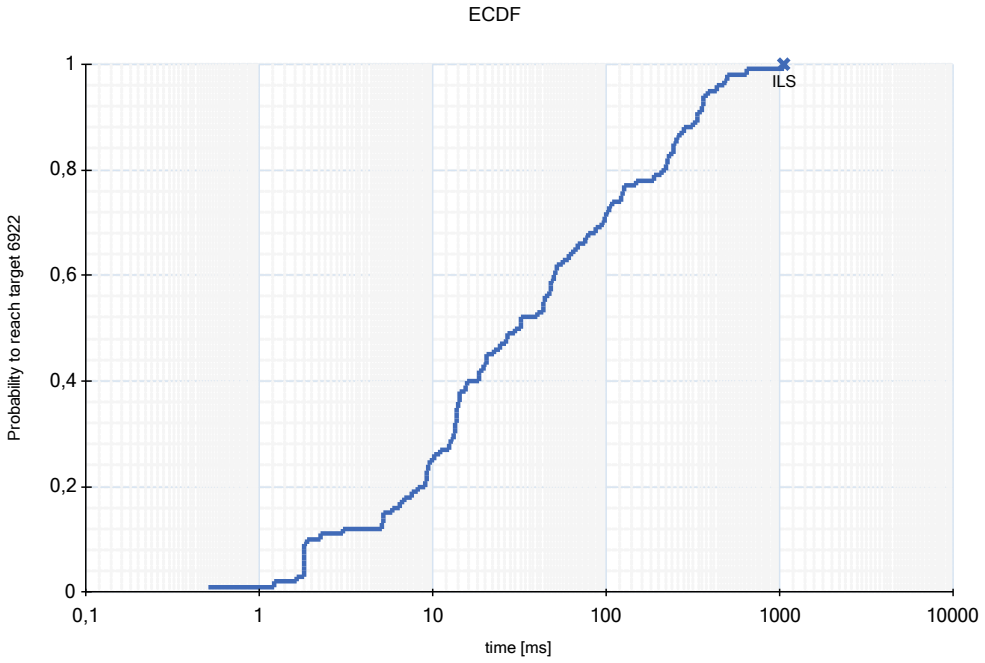



Figure 6: Empirical Cumulative Distribution Function (ECDF) generated as a result of a run-length analysis. An ILS is applied to the had20 QAPLIB instance 100 times with the target set to the global optimum.

that interpret the results. But the question of what constitutes a meaningful improvement or target has to be answered before conducting the experiment, potentially in collaboration with the creator of the problem instance.

### 2.5.2 Run-length Distribution

The idea to describe algorithm performance by treating the run-time towards a target as a random variable was present in [FRS94], but first described formally in [HS98]. They coined the term "run-length distribution" (RLD) which allows arbitrary cost models such as function evaluations to be used instead of limiting it to elapsed wall-clock time. The RLD of an algorithm instance can be computed for every pair of problem instance and target value. RLD plots show the percentage to achieve a certain target on a certain problem instance as a function of the run-length. To compute and visualize the run-length distribution several tools have been described.

**Time-to-target (TTT) Plots** have been described in [ARR07] and a perl program has been published to generate these plots. The required data for TTT plots are individual runs that record the time a certain target was reached. The script that generates these plots also fits a shifted exponential distribution to the data which approximates the observed data.

**Comparing Continuous Optimizers (Coco)** is a set of tools to conduct experiments and process results achieved on instances of the black-box-optimization-benchmark suite (BBOB) which are discussed at conferences such as GECCO or CEC [FR14].

**HeuristicLab** features a "Run-length Distribution View" which is a GUI-based tool that analyzes the convergence graphs present in *Run* objects which are created from algorithms implemented in HeuristicLab [WKB+14]. It allows to group runs by parameters, has the possibility to define the targets a posteriori and allows comparing among multiple problem instances for which a best-known value is available. This tool also calculates the expected run-time (ERT) for each instance of algorithm, problem, and target value similar to [HAFR09].

## 2.6 Algorithm Selection

The algorithm selection problem (ASP) [Ric76] was introduced to formalize the problem of choosing a set of algorithms suitable for some specific task when several such algorithms are available and not one specific algorithm outperforms all others, i.e., achieves the best result in shortest time.

Kotthoff wrote a recent survey on algorithm selection approaches, noting that while it has been shown that algorithm selection may work well in benchmark and competition scenarios, a translation to real-world application scenarios is still missing [Kot16]. It is often observed in research works that some problem instances may be favourably solved by a certain algorithm, while that same algorithm may exhibit a worse performance on a different set of instances. In the extreme cases, algorithm performance may range from extremely well to extremely poor among instances of the same problem. Thus, a major motivation for the study of fitness landscapes is the ability to relate landscape structure with algorithm performance. The goal is to detect such structures efficiently, either before or during the application of algorithms, and then decide on an effective algorithm. This, so called, algorithm portfolio together with the recommender or selector is then used to solve instances of the problem. Examples of such portfolios may be found in the works of Xu et al. [XHHLb08] or Smith-Miles et al. [SMJGT09, SMvH11].

Unfortunately, such portfolios also bear some complexity. First, there is the challenge of portfolio composition and benchmark instance selection [XHLb10, SMvH11]. Which algorithms should be part of the portfolio and which are to be excluded? Which problem instances are to be used to benchmark the algorithms? As Smith-Miles and van Hemert note "*There is nothing meaningful that will be learned from the meta-data if all instances map to the same region in feature space [..], or if all algorithms perform similarly on all instances*" [SMvH11]. Furthermore, each algorithm may have parameters, which in turn leads to a large range of possible algorithm instances. However, only fully parametrized implementations of algorithm instance may be benchmarked on a range of machines.

Second, there is the challenge of identifying a good selection model to discern the algorithm instances [SM08]. This challenge often requires that the performance of each algorithm (instance) on each problem (instance) from a training set needs to be known. This requires extensive amounts of computational resources in benchmarking the algorithm instances on the training set.

The learning effort put into such a selection model is not to be underestimated.

Third, there is the challenge of orchestration. Algorithm instances may be run independently of each other or may form a more carefully chosen schedule [PT09, GS09]. In such a schedule, an algorithm instance uses results of a previous run instead of restarting from a randomly drawn solution. In a similar way selective hyper-heuristics are to be seen as dynamically selecting from a set of heuristics and thus implicitly creating such a schedule [WS11b, BGH+13, SHP15]. Blackboard systems are a slightly different agent-based approach to hyper-heuristics [GS17] which differ in the way control over the heuristics/agents is exerted. Also the agents in blackboard systems are more capable than the heuristics and estimate their contribution to a current state of the solving process.

Several publications describe methods and software packages have been made and support researchers in the task of configuring algorithms and problem instances. First, to find a set of meaningful algorithm instances, one may either rely on human experience or on a number of published methods and implementations such as SPO Toolbox [BBLP10], ParamILS [HHLBS09], SMAC [HHLB11], GGA [AST09], or irace [LIS14, LIDLP+16]. Smith-Miles and van Hemert have described an approach to evolve a benchmark library of problem instances [SMvH11]. Xu et al. describe an approach called Hydra to combine parameter optimization and portfolio building [XHLb10].

A comparison between two variants of tabu search and two variants of simulated annealing, all with default parameter settings, has been performed [HS14]. The authors evaluated the influence of problem instance size on the rank of the best algorithm and found out that problem instance type, size, and runlength of the algorithms have an influence on which algorithm instance performs best.

In further applications, Liefooghe et al. [LDV+17] describe a study that evaluates a novel landscape-aware approach to the configuration of algorithms. They use instances from the $NK$ family of problems and cluster them using landscape features. They then apply irace to each cluster and devise optimal configurations for each cluster. Finally, they show that the optimized configurations together with a selector may achieve better solutions than a single baseline configuration optimized on all instances. The authors state that "one particularly promising idea consists in carefully choosing the instances where some configuration should race at every iteration based on the features values of the instances experimented in previous iterations" [LDV+17]. This strengthens

the argument that choosing problem instances is an important step [SMvH11]. Bozejko et al. [BGN+19] describe a study on algorithm selection present in Google's OR Tools[1] on the traveling salesperson problem (TSP). The authors conclude that of the provided choices a single algorithm outperformed all others. However, they describe that the "automatic" choice provided by OR Tools did not select that algorithm. Instead an algorithm was applied that performed worst among those tested leaving room for improvement [BGN+19]. Kotthoff et al. [KKHT15] describe a study of algorithm selection on the TSP. They apply two well-known algorithms (1) Lin-Kerninghan and (2) EAX both with a simple restart strategy that they showed improved search performance considerably. They describe that relying on the single best solver EAX+restart as a probing algorithm that also generates features for a subsequent selection step enables to make better decisions than running EAX+restart alone [KKHT15]. These results suggest that an iterative approach to solving instances in terms of a schedule of algorithms with intermediate steps of analyzing landscapes is a promising approach. Wagner et al. [WLM+18] describe a study of algorithm selection on the traveling thief problem (TTP). Finally, the well-known "algorithm selection library" (ASlib) [BKK+16] has to be mentioned; originating from the domain of satisfiability (SAT) problems it also includes a release of TSP and TTP.

However, as Kotthoff notes that "Static portfolios are necessarily limited in their flexibility and diversity." [Kot16]. Thus, the future challenge in this research domain is to create lifelong machine learning systems [SYL13, SHP15] that continuously learn and adapt using e.g. reinforcement learning methods.

[1] `https://developers.google.com/optimization/`

# 3 Exploratory Landscape Analysis for Combinatorial Problems

> "The uses or functions of a screw driver cannot be algorithmically enumerated."
>
> S. Kauffman [Kau14]

While many articles have been published on exploratory landscape analysis (ELA) in continuous search spaces, combinatorial search spaces have received less attention. A peculiarity of combinatorial search spaces in general is the difficulty of defining "direction". In continuous search spaces this concept is prevalent and exploited heavily. In this section, my intention is to describe a new kind of walk for exploratory landscape analysis, that, on the one hand combines properties of walks mentioned in the previous section, and on the other hand introduces a notion of direction along the edit-distance. These walks are named "directed walks" [BPWA17, BAW17] accordingly and will be described and analyzed extensively in this chapter. The directed walk is analyzed extensively on fitness landscapes of the quadratic assignment problem, but it is a generic method that can be applied to many other combinatorial landscapes. Directed walks require the presence of a neighborhood relation, as is mandated by the definition of a fitness landscape itself (cf. page 32). In addition, a distance function must exist with the property that for any pair of different solutions, there is at least one neighbor of the first solution with a strictly smaller distance to the second solution.

In summary, in this section the following contributions are made to the state of the art, which is also represented in the structure of the respective subsections:

- 3.1 Introduce directed walk and additional variants and analyze them on various benchmark instances of the quadratic assignment problem.
- 3.2 Introduce additional features and analyze existing features.
- 3.3 Study the performance of directed walks to identify the instances and problem classes under a varying effort and with a variety of feature sets.
- 3.4 Discuss the integration of ELA in metaheuristic algorithms.

## 3.1 Directed Walks

As has been discussed in Section 2.3 several methods exist for sampling and characterising the search space and a number of different walks has been reviewed. The disadvantages have been discussed in that each walk presents its own picture and that there is considerable sampling effort. In this section, I want to present a unifying picture by introducing a walk that aims to combine different aspects of the aforementioned walks into one walk.

The directed walk (dw) provides one sampling strategy that creates suitable characteristics. The motivation behind directed walks is that we want to explore parts of the solution space that are also relevant to optimization algorithms, while still aiming to cover the search space as a whole. In addition relevant characteristics of problem instances should be represented with a range of features.

In the directed walk, one chooses certain start and destination points. To cover a larger part of the search space, these points should have a high distance to each other, i.e. the destination should be mostly dissimilar to the starting point. The directed walk then attempts to draw a path from the starting to the destination solution which

1. strictly closes distance
2. chooses the best among the alternative neighbors

Thus, directed walks make use of a restricted neighborhood that includes only those neighbors that are closer to the destination solution. The advantage of such a neighborhood is its size; it is less than the full neighborhood and further reduces in size as the path approaches the destination. The computational complexity may thus be reduced in contrast to adaptive or up/down walks. In addition, no parameter is required that would otherwise define the sample size of the neighborhood, because this is governed by the dissimilarity of the two solutions. Furthermore, as the best among the alternative neighbors is chosen, the walk explores parts of the solution space that are of higher quality while, in contrast to adaptive walks, the bias towards these regions can be controlled quite well.

A directed walk is based on the path relinking (PR) heuristic that records its trail. In PR a trajectory is created that links two solutions in the search space [GLM00], for instance by making a greedy choice in each step. In Figure 7 an exemplary path is shown. The solutions $A$ and $B$ are initially different in 4

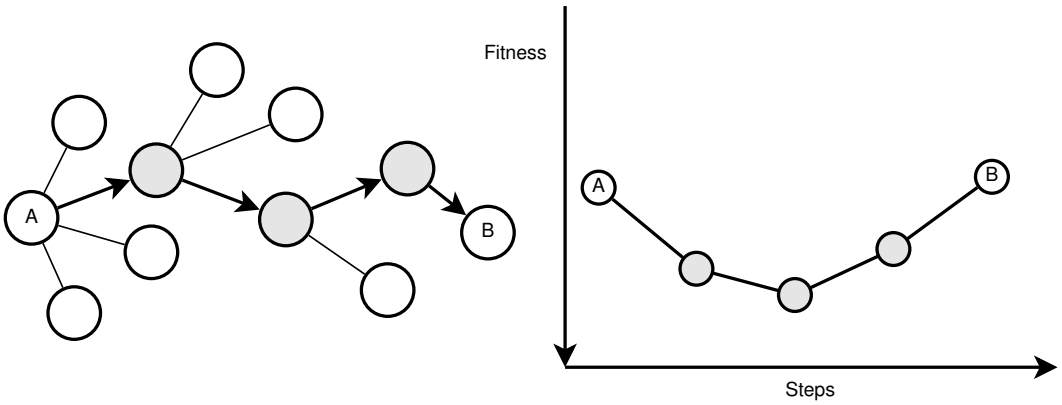



Figure 7: Path relinking between $A$ and $B$, adapted from [BPWA17].

positions. The path is represented as the three intermediate solutions that link $A$ and $B$. On the right of Figure 7, the corresponding fitness chart is shown. The $U$-shape is depicted in this chart in that the best-found solution of the path is the one farthest from both $A$ and $B$.

The use of path relinking for exploratory landscape walks in the form of directed walks is described [BAW17]. In Algorithm 4, the directed walk traces the trajectory between a starting solution $s_0$ and a target solution $s_t$ [BAW17]. It remembers visited solutions in a trail variable, i.e., a list-like data structure. Directed walks use a restricted neighborhood $N^r$ that consists only of solutions $s'$ that are more similar to the target solution $s_t$ than the current solution $s$. Naturally, this neighborhood becomes smaller and smaller with each step. The best solution from that restricted neigbhorhood then replaces $s$ before a new step may be performed. The trail is updated with each new solution and its associated fitness value. The directed walk ends when $s$ and $s_t$ are identical and the resulting trail is returned. Both solutions $s_0$ and $s_t$ are part of the trail. In the case of $s_0 = s_t$ the trail contains only one solution. A potential selection bias may be introduced in Line 10 of Algorithm 4 when the neighborhood is generated deterministically and a deterministic policy of selecting among equally fit candidates is in place. Such a bias could be resolved by randomizing the neighborhood or using a stochastic selection policy. However, even with a bias in place, the outcome should not be affected negatively.

The runtime complexity of directed walks is $\mathcal{O}(n^2)$ with $n$ being the difference between the starting and the target solution and also the amount of steps observed in the landscape per walk. This is worse than random walks which is simply $\mathcal{O}(n)$ when $n$ steps are made.

**Algorithm 4** Directed Walk

```
1: procedure DirectedWalk(↓ s_0, ↓ s_t)
2:     trail ← [ ]                                   ▷ Initialize trail to an empty list
3:     s ← s_0
4:     while true do
5:         trail ←+ (s, fitness(s))
                                          ▷ Calculate the restricted neighborhood N^r
6:         N^r ← {s' ∈ N(s) | distance(s', s_t) < distance(s, s_t)}
7:         if N^r = ∅ then
8:             return trail
9:         end if
                                  ▷ Choose the neighboring solution with best fitness
10:        s ← arg min_{s*∈N^r} fitness(s*)
11:    end while
12: end procedure
```

Generally, a relaxation of directed walks can be thought of in terms of the restricted neighborhood. Given the definition in Algorithm 4 Line 6 the neighboorhood consists only of neighbors that strictly close the distance to the target solution $s_t$. A relaxed neighborhood would include neighbors that maintain the same distance. However, in this case we may add additional requirements, for instance that such a "distant neutral move" is also strictly improving in terms of fitness. We will take a look at using such a relaxation later in this section when we analyze inverse directed walks.

In the following three variants of directed walks are discussed that differ in the type of points used as start and destination. In Section 3.1.1 we analyze directed walks between randomly generated solutions. I use the notation *(rr)-dw* to denote this variant, which reads: (from randomly sampled to randomly sampled solution)-directed walk. In Section 3.1.2 directed walks between randomly sampled solutions and a global optimum are analyzed, denoted as *(rg)-dw* and in Section 3.1.3 directed walks among local optimal solutions are analyzed, denoted as *(ll)-dw*. In Section 3.1.4 we will analyze directed walks between local optima and a gobal optimum, denoted as *(lg)-dw* and finally, in Section 3.1.5 we will look at the aforementioned inverse directed walk variant termed *(li)-dw*. Of course, further variants and analysis can be thought of using different combinations of points. In Section 3.1.6 we will compare directed walks to other existing walks.

### 3.1.1 Directed Walks between Random Solutions

Figure 8 shows several trajectories between such randomly generated solutions in terms of the fitness progress. As the QAP objective is to be minimized, the trajectory is described by a $U$-shape. In the plot in Figure 8 the y-axis shows solution quality and the x-axis shows intermediate solutions as they are sampled along the trajectory. In the beginning, the restricted neighborhood is still fairly large and there is likely room for most improvement. Then, usually a trough follows in which the neighborhood becomes smaller and less improvement is possible. As the intermediate solutions become more similar to the destination, quality degrades. More and more of the bad solution components of the randomly generated target solution have to be added. Finally, it reaches the destination's fitness which is again of similar quality to the starting solutions given that both are sampled alike.

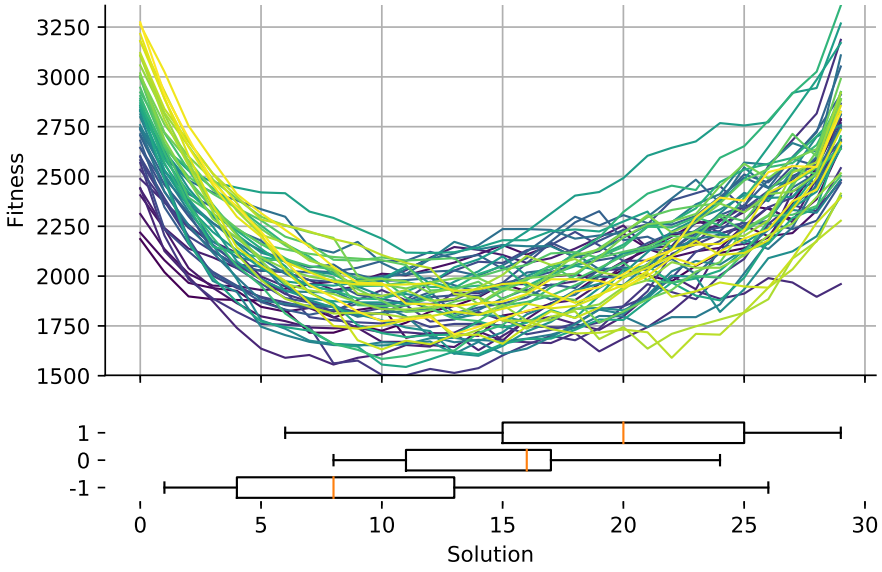



Figure 8: The progress of 50 directed walks between randomly generated solutions[2]on the *dre30* instance. Below the frequencies of downward (-1), neutral (0) and upward (1) slopes are shown.

Another nice property of directed walks is that they can be overlaid as was done in Figure 8. The overlay still shows essentially the same pattern as the individual walks. Compared to random walks from randomly sampled

[2]A biased Fisher-Yates shuffling algorithm is used to create two solutions that do not share a common element and are thus of maximum distance to each other.

solutions, the overlay would be unlikely to reveal any sort of structure except for a constant band of best and worst observed fitness in which all the individual walks wiggle up and down. Directed walks enable us to create an 'average walk' out of the individual walks and obtain characteristics from this smoothed combination. A downside here however, is that each path should be of similar length. This property holds mostly when two randomly generated solutions are used. The color in Figure 8 and the following figures has been scaled depending on the quality of the starting solution. We can observe that for the instance shown there is no visible relation between the initial quality and the best quality in such walks.

When comparing directed walks among different instances, various patterns emerge. In Figure 9 two instances from the QAPLIB are compared in terms of their directed walks. The *esc32f* instance has a large neutral part which stems from the fact that it contains several "dummy" facilities that do not have weights. Reassigning those dummies to different locations does not affect the objective. In contrast, the *tai30a* instance is randomly generated and does not have these dummies. It doesn't show any amount of neutrality. The plots reveal these properties very well as can be observed by comparing the occurrence of neutral moves in Figure 9.

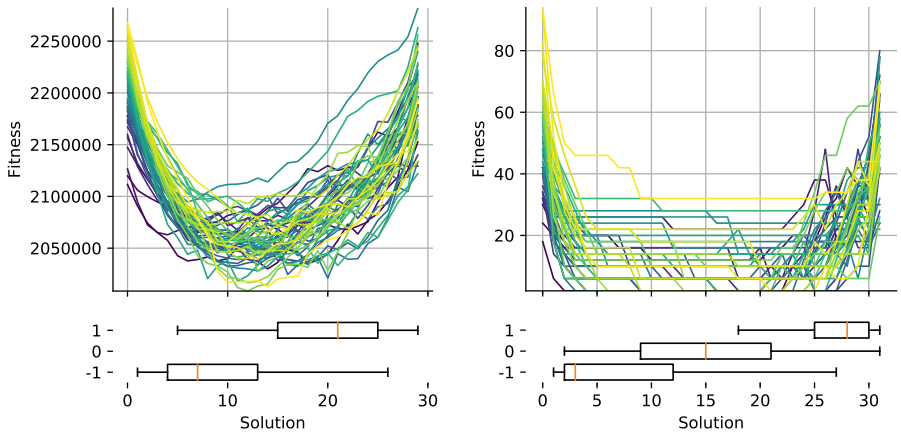



Figure 9: The progress of 50 directed walks between randomly generated solutions on the *tai30a* (left) and *esc32f* (right) instance (QAPLIB). Below the frequencies of downward (-1), neutral (0) and upward (1) slopes are shown.

### 3.1.2 Directed Walks between Random and Optimal Solutions

However, analyzing paths between randomly selected solutions is but one possibility of applying directed walks. Other, potentially interesting points in the landscape may act as starting or destination points for directed walks, revealing different characteristics. In Figure 10 the trajectories between randomly generated solutions and an optimal solution are shown. For this problem instance progress towards the optimum solution is inhibited at about step 20 when no improving move leads closer to the optimum solution. In such a situation a greedy search would either be in a local optimum or directed away towards a more distant local optimum. We can hypothesize that such analysis would enable us to capture deceptiveness of a problem. A lot of degrading moves in these walks would indicate that the progress towards an optimal solution cannot rely on the "local gradient", i.e. an improving move in the neighborhood. In the opposite case, mostly down-hill moves would indicate that global optimal solutions can be attained by following improving moves to some degree.

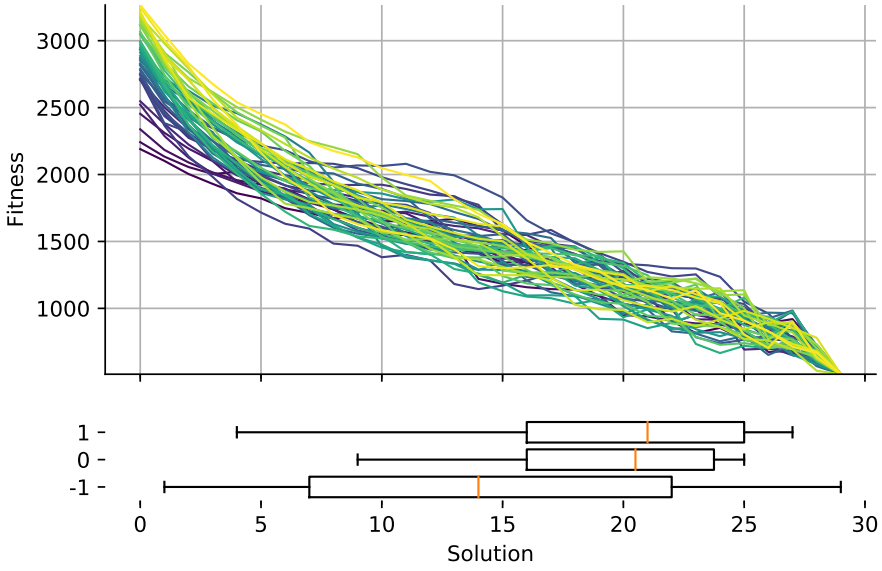



Figure 10: The progress of 50 directed walks from a randomly generated solutions towards an optimum on the *dre30* instance. Below the frequencies of downward (-1), neutral (0) and upward (1) slopes are shown.

Again, comparing directed walks among problem instances reveals the instances' characteristics. In Figure 11 convergence for the *esc32f* is achieved

much faster, followed by many neutral changes and degrading moves being delayed than in the *tai30a* instance where a gradual progress can be observed that does not involve neutral moves.

Of course, analysis based on optimal solutions are only possible in benchmark situations when the problem instances have been solved to optimality. At this point, naturally, the question emerges if this is useful as it cannot be applied to new problem instances. It may be useful when the goal is to understand fitness landscapes better, making heavy use of already well-studied instances. However, when our goal is to characterize new and previously unseen instances, an analysis based on the global optimum is inhibiting. To resolve this situation, a similar analysis can be made on local optima which are easier to achieve, even for new instances, and which shall reveal similar effects. Nevertheless, it has to be mentioned, that the effect is likely less pronounced when the landscape consists of many local optima and the target local optimum is of bad quality.

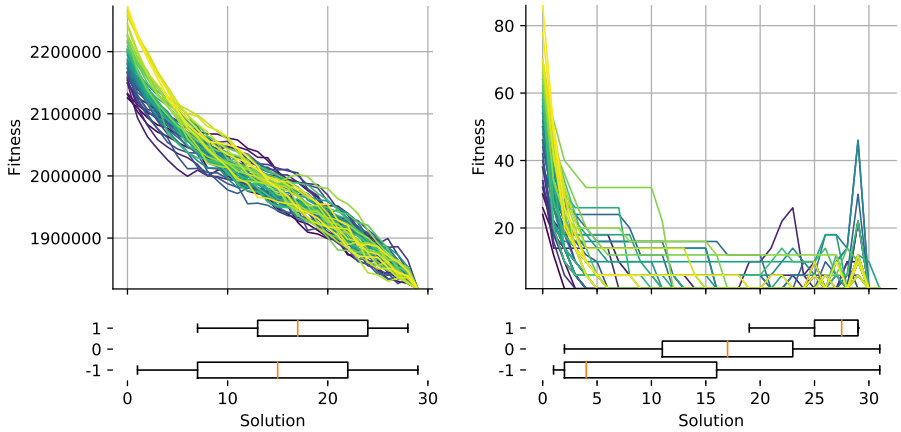



Figure 11: The progress of 50 directed walks from randomly generated solutions towards an optimum on the *tai30a* (left) and *esc32f* (right) instance (QAPLIB). Below the frequencies of downward (-1), neutral (0) and upward (1) slopes are shown.

### 3.1.3 Directed Walks between Local Optimal Solutions

Another way of analyzing landscapes is by exploring the paths between local optimal solutions using directed walks. The "U-shape" that was observed in walks between randomly generated solutions is flipped upside down for these

kinds of walks. Nevertheless, it is not a priori known to which degree the paths degrade in fitness. Potentially, this reveals aspects on the steepness of local optima in the landscape. Because local optima are also valuable solutions, the analysis of the landscape coincides to some degree with the intent of finding good solutions.

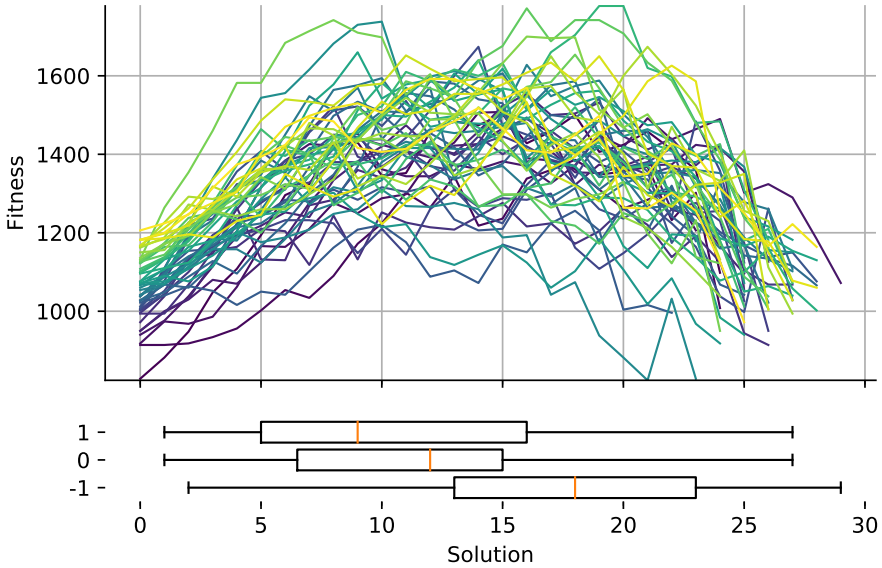



Figure 12: The progress of 50 directed walks between local optimal solutions on the *dre30* instance. Below the frequencies of downward (-1), neutral (0) and upward (1) slopes are shown.

In Figure 12 the paths of directed walks are displayed between two local optima. The optima have been computed using best-improvement local search in the *swap2* neighborhood. Each local optimum, except for the first and the last were start and destination in at least two paths. It is rather difficult to generate an adequate number of local optimum of maximum distance to each other, thus in this figure some paths terminate before others. It is visible that only a few of the paths remain well within the range of local optima as determined by the fitness distribution in the starting points. Most paths have to cross parts of the search space that are much worse than that.

In Figure 13 a comparison between the *tai30a* and the *esc32f* is performed. Again, the absence of neutrality in the *tai30a* is shown in that there are only improving or degrading moves out of those that are interesting, i.e. best alternatives in the path. For the *esc32f* we can observe a large number of neutral steps, which probably corresponds to the alignment of all "dummy" facilities,

before a rather wiggly amount of up- and down-hill moves as well as some neutral moves still need to be made.

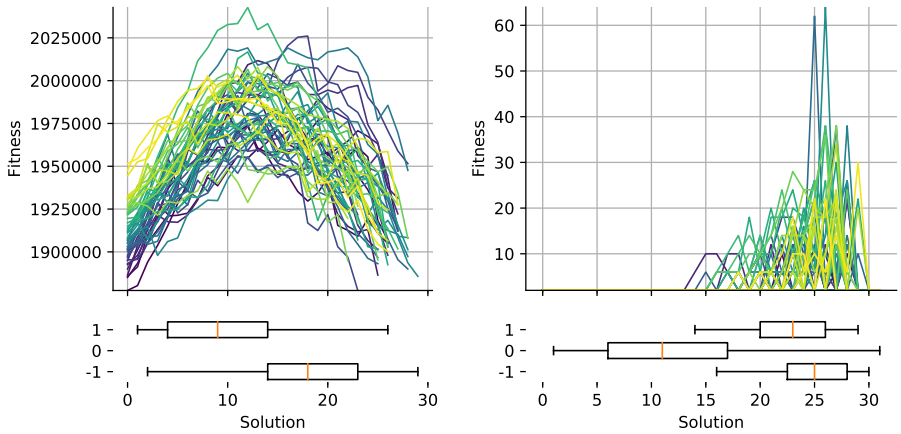



Figure 13: The progress of 50 directed walks between local optima on the *tai30a* (left) and *esc32f* (right) instance (QAPLIB). Below the frequencies of downward (-1), neutral (0) and upward (1) slopes are shown.

### 3.1.4 Directed Walks between Local and Global Optimal Solutions

Finally, in the last analysis of directed walk variants in this chapter, local optima shall be used as starting points and the global optimum is chosen as destination. This will present insights into the relation among the, arguably, most interesting points in a fitness landscape. In general, if local optima are clustered around global optima we would expect to see shorter paths as these two points share some similarity. On the other hand, in the amount of degrading moves, we can hypothesize on, e.g. the strength of the perturbation that we need to made in order to find the global optimum using heuristic techniques.

In Figure 14 the paths of directed walks are displayed between two local optima. The optima have been computed using best-improvement local search in the *swap2* neighborhood. It is visible that only a few of the paths remain well within the range of local optima as determined by the fitness distribution in the starting points. Most paths have to cross parts of the search space that are much worse than that.

In Figure 15 a comparison between the *tai30a* and the *esc32f* is performed. Again, the absence of neutrality in the *tai30a* is shown in that there are only

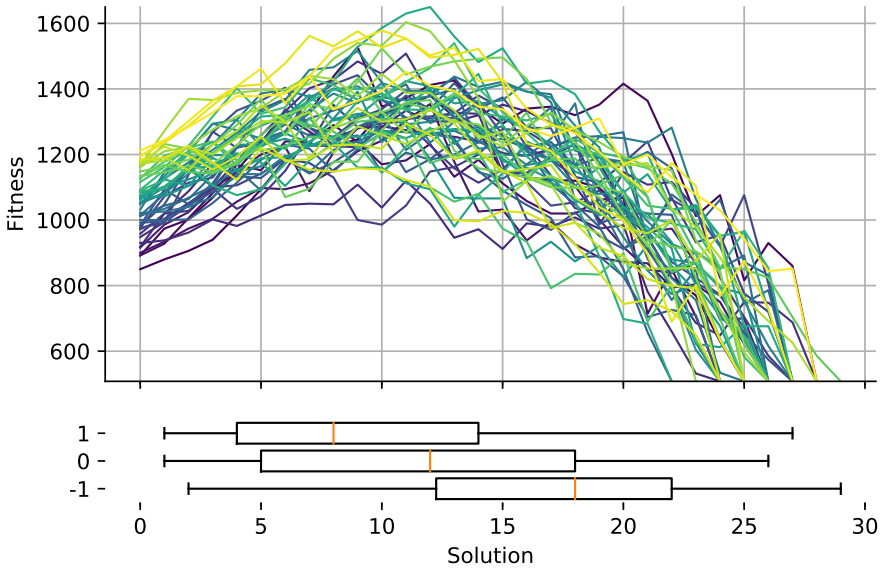



Figure 14: The progress of 50 directed walks between local optima and a global optimum on the *dre30* instance. Below the frequencies of downward (-1), neutral (0) and upward (1) slopes are shown.

improving or degrading moves out of those that are interesting, i.e. best alternatives in the path. For the *esc32f* we can observe a large number of neutral steps, which probably corresponds to the alignment of all "dummy" facilities, before a rather wiggly amount of up- and down-hill moves as well as some neutral moves still need to be made.

### 3.1.5 Inverse Directed Walks

In directed walks the goal was to find a path between two, mostly dissimilar, solutions that connect each other in a best-improvement way. In inverse directed walks, we would instead take one solution with interesting properties (e.g. locally optimal) and use it to direct the walk away from it under a best-improvement sampling strategy. We abbreviate this variant as *(li)-dw*. Thus, the walk is constructed by step-wise removal of all components of the starting solution. The walk stops when a solution of maximum dissimilarity has been reached, which in the case of the QAP takes at most $N - 1$ steps. The algorithm description of inverse directed walks is similar to that of directed walks in Algorithm 4, but inverses the inequality as shown in Equation (3.1). A schematic example of such a walk is shown in Figure 16.

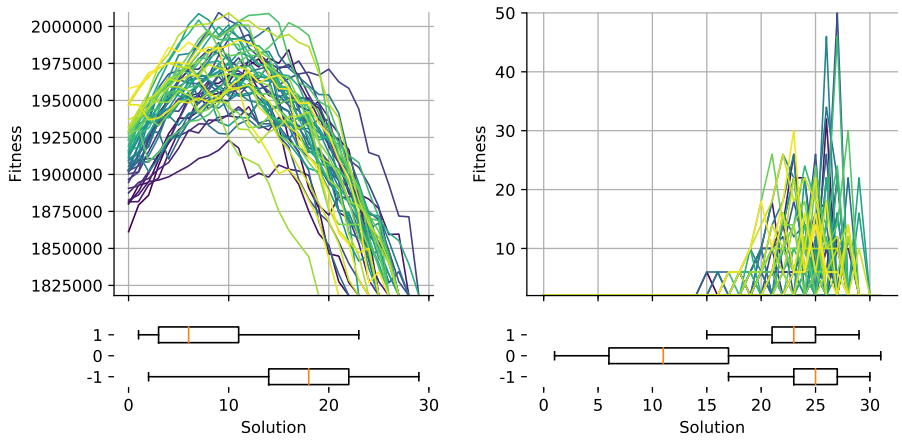



Figure 15: The progress of 50 directed walks between local optima and a global optimum on the *tai30a* (left) and *esc32f* (right) instance (QAPLIB). Below the frequencies of downward (-1), neutral (0) and upward (1) slopes are shown.

$$N^r \leftarrow \{s' \in N(s) \,|\, dist(s', s_0) > dist(s, s_0)\} \tag{3.1}$$

Directed walks and also its inverse variant are deterministic algorithms. The only source of randomness is the starting solution, respectively also the destination solution in case of regular directed walks. However, these are input to the algorithm and no further stochastic choices are made. A consequence of determinism is that there exists only exactly one walk to every solution. But as there is a form of memory in terms of the starting solution, two walks that cross each other on the same solution may not continue identically.

However, inverse directed walks would also be possible "between" two solutions. One solution acts as the starting solution, while another acts as a "repelling" solution. The walk is then confined to keep the unique components of the starting solution while performing a stepwise replacement of all components that are shared with the repelling solution. Thus, an inverse directed walk can be similarly confined to a sub-space of the search space governed by two solutions.

In Figure 17 inverse directed walk paths are shown. The characteristic behavior that can be observed is that walks are able to improve on the local optima in the first couple of steps, but then gradually degrade in quality as more and more good properties of the solution have to be replaced by alternatives. The possibility of improving on the local search quality in such a deterministic extension of local search has not yet been explored very much

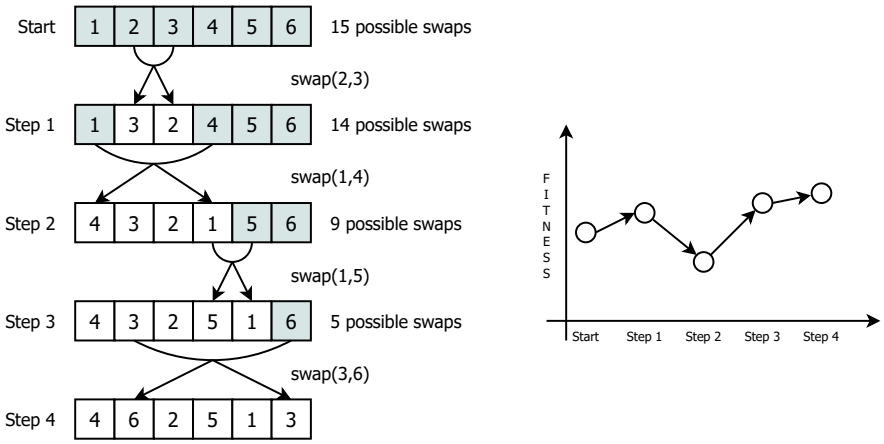



Figure 16: A schematic example of an inverse directed walk of a discrete permutation-based encoding with a hypothetical fitness progress on the right.

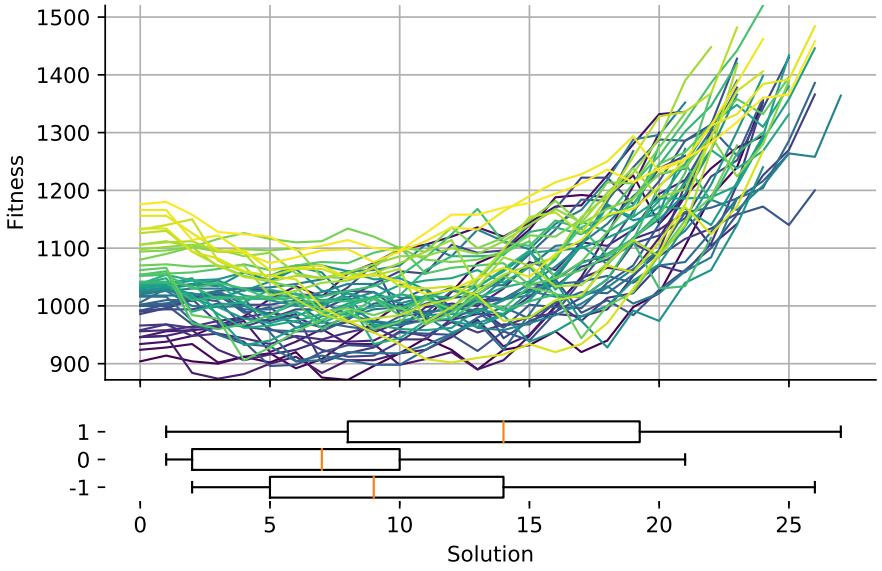



Figure 17: The progress of 50 inverse directed walks starting at local optima on the *dre30* instance. Below the frequencies of downward (-1), neutral (0) and upward (1) slopes are shown.

in the literature. Typically, a local search is performed and terminated once a local optima has been reached. However, as Figure 17 shows, a simple inverse directed walk could provide an additional improvement over such local optima in a few steps. In comparison with Figure 12 we observed that the path *between* local optima involves the "crossing of a hill". Thus, improving solutions to the QAP by means of an exploration path between local optima is rather difficult. However, exploring the basin around local optima by means of inverse directed paths may be fruitful.

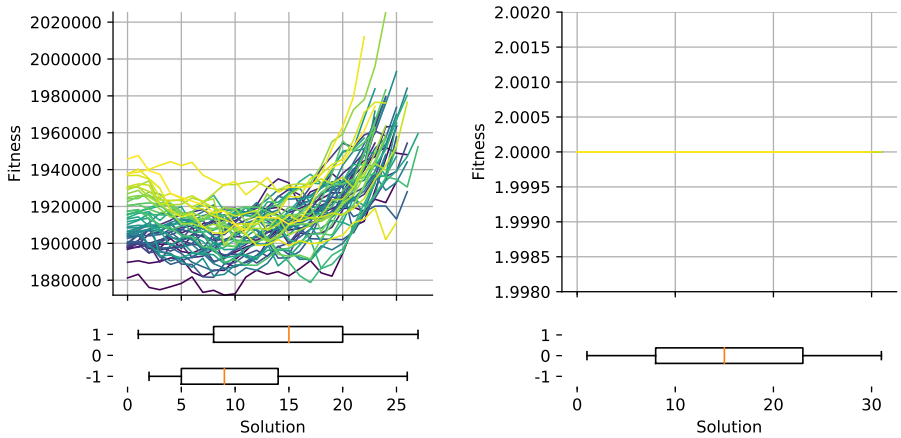



Figure 18: The progress of 50 inverse directed walks starting at local optima on the *tai30a* (left) and *esc32f* (right) instance (QAPLIB). Below the frequencies of downward (-1), neutral (0) and upward (1) slopes are shown.

In Figure 18 inverse directed walks are compared again between a random instance and one showing a high amount of neutrality. The differences are very obvious in that the high amount of neutrality enabled the walk to remain non-improving and non-degrading, i.e. a flat line. The random instance again does not show any signs of neutrality and degrades very quickly in quality towards the end. The behavior is similar to that observed for the *dre30* instance shown in Figure 17.

**Relaxed Inverse Directed Walks**

As has been described in the beginning of Section 3.1 a relaxed variant of inverse directed walk can be created by modifying the requirement that distance to the starting solution must increase with every step. Instead we would require, that distance in every step must be *non-decreasing*. This includes changes in those components that already differ from the starting solution.

However, such a walk could run endlessly. Thus it is also required that moves, that remain at the same distance, must also be strictly improving. If there is no further move to increase distance and no further improving move in this relaxed neighborhood, than that solution would be "distant-locally optimal". Note however, that such solutions may be rather bad. For instance, considering the case when local (and global) optima are clustered in the search space and thus have several components in common. The resulting "distant-locally optimal" solutions could, by definition, not contain these shared components. Thus, their fitness would be worse.

If the starting solution and the global optimal solution do not share any components, then the global optimum could theoretically still be reached by such a walk. Such a setting might be more successful in cases where the landscape is considered to be "deceptive" to some degree. Algorithmically, the relaxed variant of inverse directed walks is similar to the aforementioned variants, but differs in the calculation of the neighborhood $N^r$ as given below. The disjunction in Equation (3.3) allows additional neighbors.

$$N^r \leftarrow \{s' \in N(s) \mid dist(s', s_0) > dist(s, s_0) \tag{3.2}$$

$$\vee\ [\, dist(s', s_0) = dist(s, s_0) \wedge fit(s') < fit(s)\,]\,\} \tag{3.3}$$

### 3.1.6 Relation of Directed Walks to Other Existing Walks

As has been said, directed walks aim to combine many attributes of existing walks, such as exploring better and worse regions of the solution space and because of its greedy nature also mimic some of the behavior that would likely be seen by an optimization algorithm. In the following a brief comparison of directed walks to previously described walks is performed:

**Directed vs Up/Down walks**: Directed walks explore such up and down paths in any variant due to the greedy selection in the restricted neighborhood. The *(rg)-dw* and potentially *(rl)-dw* variant focus explicitly on down walks. But *(rr)-dw* and *(ll)-dw* explore into better and worse regions of the search space. Directed walks thus feature a similar descending and ascending behavior to up/down walks. Some of the described features, such as *up (down) walk length* could also be computed from directed walks.

**Directed vs Adaptive walks**: Both adaptive and directed walks greedily choose the neighbor in each step. However, adaptive walks use an unbiased random sub-sampling of the neighborhood, while directed walks use a struc-

tured approach. The advantage is that the amount of sub-sampling need not be determined a priori, but the disadvantage is that, e.g. in *(rr)-dw* the extent to which good parts of the search can be explored is limited. Determining the amount of sub-sampling however is crucial for adaptive walks. An amount too low would equal a random walk, while an amount too high could limit the walks ability to explore the search space. If the neighborhood would not be sub-sampled, adaptive walks would be identical to local search and thus be easily trapped in a local optimum. Thus, we can argue that characteristic that surfaces in adaptive walks should also be present in directed walks with the additional benefit that it is parameterless.

**Directed vs Neutral walks**: In a landscape with many plateaus, directed walks will also move along the plateau for some part of their walk. This has already been shown in some of the previous figures. An *(rr)-dw* will favor the plateau over an up-hill move in case only non-improving neighbors are available. The advantage being that directed walks do not look for plateaus within solutions of average quality (i.e. the starting points of neutral walks), but where these appear, e.g. near local optima in *(ll)-dw*.

**Directed vs Random walks**: Random and directed walks do not share a lot of common properties. Directed walks do move between randomly selected start points, but in an adaptive respectively greedy manner. Thus, a random walk would provide a different and unbiased view of the landscape, and complements the analysis using directed walks.

**Directed vs Progressive walks**: Directed walks make the notion of direction available in any search space. The directional vector is obtained in form of the differences in solution components, i.e. the *edit-distance*. Thus, directed walks are not confined to continuous problems and can be used in many search spaces.

## 3.2 Landscape Characteristics from Directed Walks

The quality trails that are produced by directed walks may be analyzed in a similar way to, for instance, random or adaptive walks. The difference with directed walks is their length. Directed walks typically have a certain length and do not run arbitrarily long. We can of course, concatenate the trails to form one big trail, however, we did observe one repetitive pattern in each variant of the walk in the previous sections. It may thus be meaningful to consider the average or median of these individual walks to provide descriptive characteristics of the respective landscape.

In the following several features are proposed that can be obtained from the variants of directed walks. These features will then be used later in the thesis in order to function as characteristics for algorithm selection or otherwise describe problem instances' similarities with each other. We can use these features and project the respective problem instances to low-dimensional spaces which enables us to visualize the space of problem characteristics. Such visualizations will be introduced and studied in Section 4.

### 3.2.1 Curve Analysis

In analysing the curves of directed walks we have come up with three new characteristics These are based on calculating the differential of these curves, i.e. $\frac{\Delta y}{\Delta x}$. In this case $\Delta y$ is the difference in fitness while $\Delta x$ is the difference in solution similarity or "step size". Three features have been proposed to describe these trails [BPWA17, BAW17]: Sharpness, bumpiness, flatness. In the following we will introduce these features more formally. The mathematical notation used in the description is given in Table 4. The idea is that the trails we obtain from directed walks may be very characteristic in that for a smooth landscape we would expect to see a very smooth characteristic, whereas for rugged landscapes the slope changes often. In neutral landscapes, as we have seen before, often the case is that the fitness remains constant and thus, not only does the slope stay constant, it is also horizontal.

In general, we assume the "gradient" of a certain step is given in terms of the slope induced by the previous and next steps. As shown in Figure 19 the gradient of $r$ is approximated as given in Equation (3.4). It is calculated for a step $r$ using the preceding step $q$ and the succeeding step $s$ and defined only when $q, s \neq \{\}$.

Table 4: Mathematical notation for describing curve analysis based features.

| | |
|---|---|
| **Indices** | |
| $t$ | Trajectory |
| $q, r, s$ | Steps |
| **Functions and Relations** | |
| $\text{fit}(r)$ | Fitness of step $r$, $\text{fit} : \mathcal{S} \to \mathbb{R}$ |
| $\text{agrad}(r)$ | Approximated gradient of step $r$, $\text{agrad} : \mathcal{S} \to \mathbb{R}$ |
| $\text{agrad}^2(r)$ | Approximated change of gradient of step $r$, $\text{agrad}^2 : \mathcal{S} \to \mathbb{R}$ |
| $\text{len}(t)$ | Length of trajectory $t$, $\text{len} : \mathcal{T} \to \mathbb{N}$ |
| $dx(s, r)$ | Distance (=step size) between step $s$ and step $r$, $dx : \mathcal{S}^2 \to \mathbb{R}$ |
| $dy(s, r)$ | Fitness difference of step $s$ and step $r$, $dy : \mathcal{S}^2 \to \mathbb{R}$ |
| $\text{pre}(r)$ | Returns the predecessor of $r$, $\text{pre} : \mathcal{S} \to \mathcal{S}$ |
| $\text{suc}(r)$ | Returns the successor of $r$, $\text{suc} : \mathcal{S} \to \mathcal{S}$ |
| **Sets and Sequences** | |
| $\mathcal{T}$ | Set of trajectories $t \in \{1, 2, ..., T\}$ |
| $\mathcal{S}$ | Set of steps |
| $\mathcal{S}_t$ | Sequence of steps $r_i, i \in \{1, 2, ..., \text{len}(t)\}$ in trajectory $t$, $\mathcal{S}_t \subset \mathcal{S}$ |
| **Conventions** | |
| True $\widehat{=} 1$ | A true expression corresponds to an integer of 1 |
| False $\widehat{=} 0$ | A false expression corresponds to an integer of 0 |

$$\text{agrad}(r) = \frac{dy(\text{pre}(r), \text{suc}(r))}{dx(\text{pre}(r), \text{suc}(r))} \tag{3.4}$$

$$\text{agrad}^2(r) = \frac{\text{agrad}(\text{suc}(r)) - \text{agrad}(\text{pre}(r))}{dx(\text{pre}(r), \text{suc}(r))} \tag{3.5}$$

*Sharpness* is the average absolute "gradient" in the quality trail, i.e. the ratio of the absolute quality difference and the distance between two succeeding solutions in the trail. This feature should be normalized by a best and worst quality.

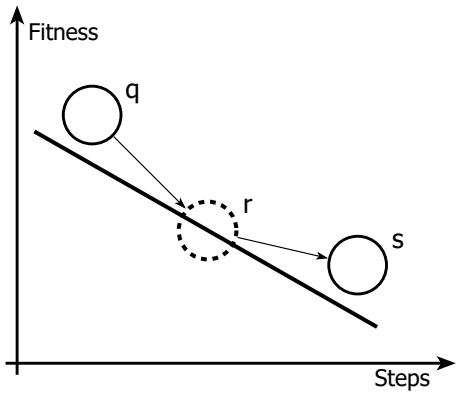



Figure 19: Visualization of the approximated gradient in a directed walk

$$\text{Sharpness} = \frac{1}{T} \cdot \sum_{t \in \mathcal{T}} \frac{1}{\text{len}(t) - 2} \cdot \sum_{i=2}^{\text{len}(t)-1} |\text{agrad}(r_i)| \tag{3.6}$$

*Bumpiness* is the relative number of "inflection points" to the points visited, i.e. where the "gradient" changes sign, but is not equal to 0.

$$\begin{aligned} \text{Bumpiness} = \frac{1}{T} \cdot \sum_{t \in \mathcal{T}} \frac{1}{\text{len}(t) - 5} \cdot \sum_{i=3}^{\text{len}(t)-3} \big[ & \text{sgn}(\text{agrad}^2(r_i)) \neq \text{sgn}(\text{agrad}^2(r_{i+1})) \\ & \wedge \text{sgn}(\text{agrad}^2(r_i)) \neq 0 \\ & \wedge \text{sgn}(\text{agrad}^2(r_{i+1})) \neq 0 \,] \end{aligned} \tag{3.7}$$

*Flatness* is the relative number of "undulation points" to the points visited, i.e. the gradient itself as well as its "derivative" are both 0.

$$\text{Flatness} = \frac{1}{T} \cdot \sum_{t \in \mathcal{T}} \frac{1}{\text{len}(t) - 5} \cdot \sum_{i=3}^{\text{len}(t)-3} \big[\text{agrad}^2(r_i) = 0\big] \tag{3.8}$$

These three features describe different aspects of the observed curves and thus of the underlying problem instances. Especially, flatness is useful to discern problem instances with a high amount of neutrality.

### 3.2.2 Information Analysis

In addition previously described features such as those described by Vassilev et al. [VFM00] may also be computed for directed walks. These features are

information content (*ic*), partial information content (*pic*), and density basin information (*dbi*) as well as peak values of information content and density basin information respectively (*ic**, *dbi**). In Figure 20 a comparison of information analysis features is given. The instances shown are *dre30* (left), *tai30a* (middle), and *esc32f* (right). The figures depict results of an information analysis from random-to-random directed walk (rr)-dw (first row), random-to-global directed walk (rg)-dw (second row), local optimum-to-local optimum directed walk (ll)-dw (third row), and local optimum-to-global optimum (lg)-dw (fourth row).

Again, it is easy to distinguish *esc32f* from the other two, e.g. based on information content alone. However, despite the notable differences in the fitness landscapes of *dre30* and *tai30a* the information analysis does not yield a comparable notable change at first sight. As we have seen in Figures 8 and 10 *dre30* did contain neutral areas in the landscape, while a complete absence of neutrality in *tai30a* could be observed in Figures 9 and 11. Nevertheless, at a closer look it can be observed that *ic* is less for *tai30a* than for *dre30* when $\varepsilon = 0$ in both the (rr)-dw and the (rg)-dw. Also, for (rg)-dw the differences of *pci* and *dbi* between the two problem instances are more pronounced than for (rr)-dw. Among all the performed variants, the differences between *tai30a* and *dre30* are most pronounced in (lg)-dw where even the peaks of the ic curves are rather different. Unfortunately, analysis using the global optimum are only of theoretical interest as the optimal solution is known a priori only for well studied problem instances. Nevertheless, we may hypothesize that the fitness landscapes between these instances differ to a larger degree when approaching the global optimum and are otherwise somewhat similar.

Due to the stochastic nature of the sampling process in exploratory landscape analysis, the obtained features are random variables. Ideally, the expected value of that variable is the same regardless of the length of the walk. Longer walks should be able to estimate the expected value to a better precision as the standard deviation drops. In practice however, we have observed that shorter walks introduce a notable bias that leads to a different distribution of the feature variable in comparison to longer walks. Certainly, it has to be mentioned that such a bias exists mostly for very short walks, but nevertheless constitutes an undesirable behavior. The box plot in Figure 21 shows the bias for different walk lengths from $2^7$ to $2^{18}$.

One issue with information analysis features roots in the symmetry of certain symbols. A random walk is typically not biased in terms of a direction.

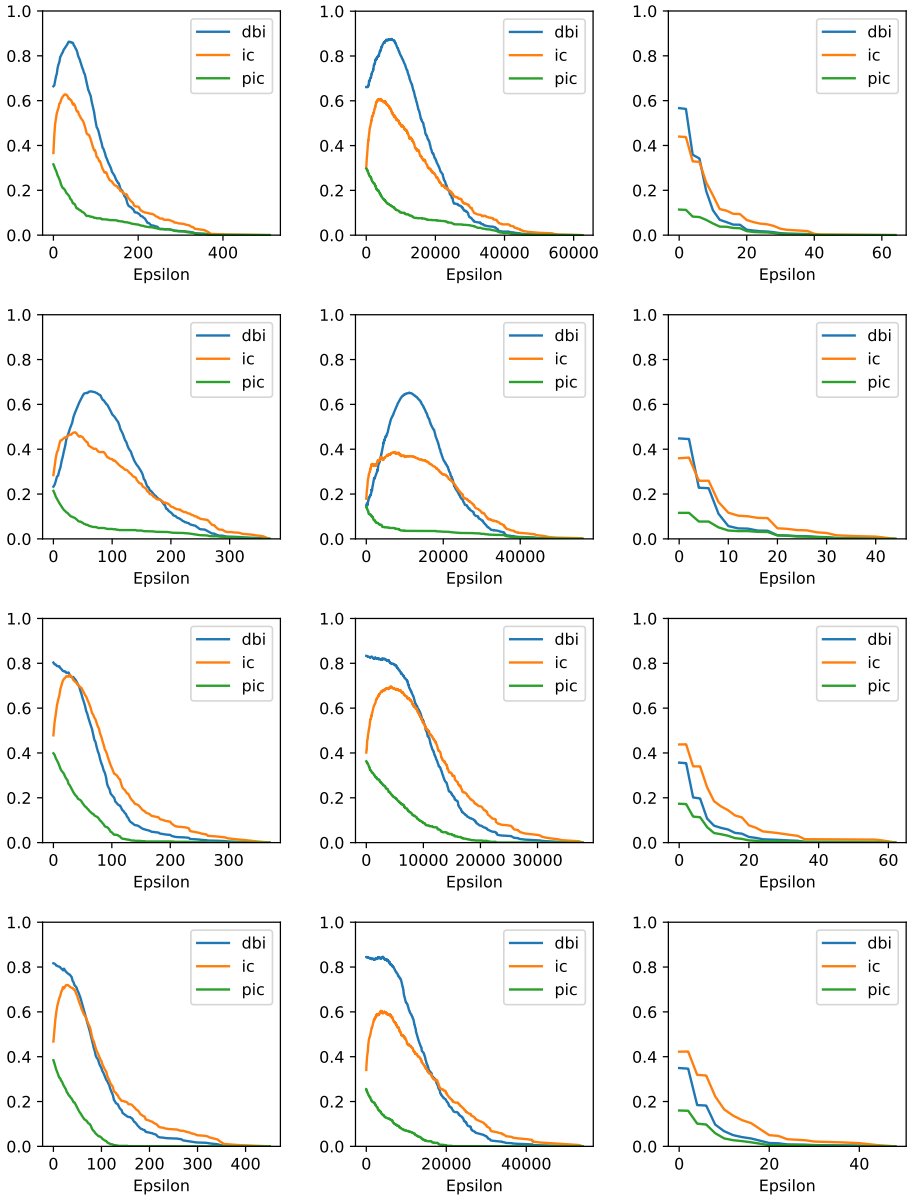



Figure 20: The result of an information analysis of directed walks applied to the QAP problem instances, from left to right: *dre30*, *tai30a*, *esc32f*. From top to bottom, the information analysis of four types of directed walks are shown: (rr)-dw, (rg)-dw, (ll)-dw, (lg)-dw.

It moves from one solution to any of its neighbors with equal probability. Thus, differences in the frequencies of symmetric pairs of symbols are due to the direction in which the walk progresses and not due to actual landscape properties. For any symmetric pair, both symbols should occur with equal probability. If the walk happened to move in the respective other direction by chance we would observe the corresponding symbol. In that case, any observed difference in those symbols is thus rather an issue of sampling inaccuracies and not due to landscape properties. The symmetric symbol pairs are ($\searrow\searrow$, $\nearrow\nearrow$), ($\searrow\rightarrow$, $\rightarrow\nearrow$), as well as ($\nearrow\rightarrow$, $\rightarrow\searrow$). Thus, instead of 9 original symbols we may only consider 3 symbols that are self-symmetric ($\searrow\nearrow$, $\nearrow\searrow$, $\rightarrow\rightarrow$) and the 3 symmetric pairs mentioned above. Figure 21 shows the *symmetric information content*. This and the symmetric density basin information are computed similarly to Equation (2.18) and (2.19) but using less symbols. The observed frequencies of the symmetric symbol pairs must be averaged before computing entropic measures, or otherwise another bias is introduced. The comparison between regular and symmetric information content is given for different walk lengths in Figure 21. For each walk length the boxplot is generated by calculating the features in 100 different runs. Each run is of length $2^{18}$ and only the first $2^i$ values are taken where $i$ is shown on the $x$-axis. The symmetric IC often gives better averages with respect to short walks than the regular IC.

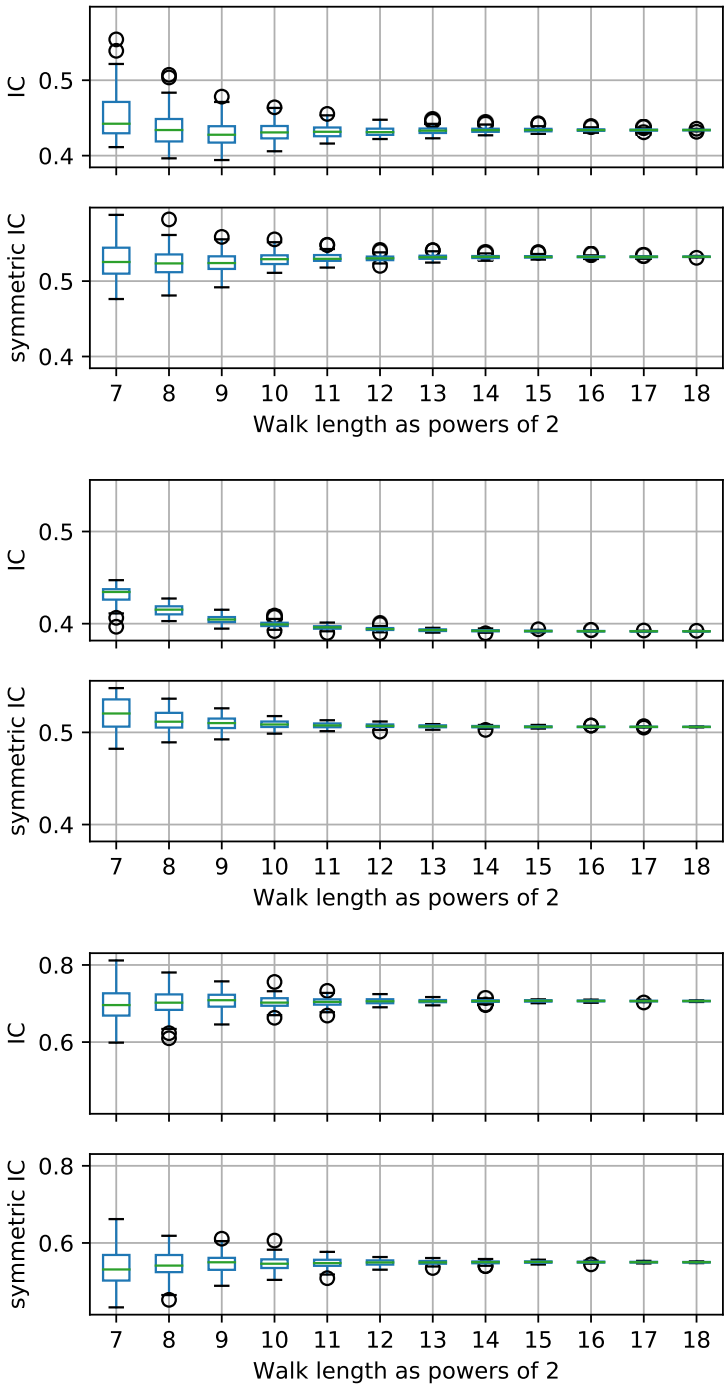



Figure 21: Comparison of regular and symmetric information content for *dre30*, *tai30a*, *esc32f*.

## 3.3 Application of Exploratory Analysis

In this section we will evaluate the application of directed walks on instances of the quadratic assignment problem. First, we will consider the stability of the features in terms of their distribution among various walks. Then we will proceed to evaluate how many walks should be made to gain a reliable estimate. Next, we will calculate the features for a certain set of problem instances two times with varying effort. The first time we produce the training data while the second time we produce the test data.

### 3.3.1 Evaluation of Feature Stability

In Table 5 several features are listed as obtained from a random walk of length $2^{18}$ in the `swap2` neighborhood. The problem instances are sampled randomly from QAP libraries described earlier in this thesis. They are reduced to a dimension of 30 by randomly removing rows and their corresponding columns from the matrices, thereby ignoring the influence of problem dimension in the analysis. Despite a wide variety of problem instances, some features result in a very low variation as determined by the coefficient of variation ($\sigma/\mu$). Given that the random walk is a stochastic procedure the exact same value cannot be computed for each feature in each run. For instance, the `AC1` - the autocorrelation given a shift of one step - is very similar for any instance. The statistical correlation length [Hor96] on the other hand shows a lot of variation, but also within problem classes. Consider instances of the *lipaXXb* or *dre* generators. The density basin information (`dbi`) also shows little variation, but this can be explained given that there are less instances with a lot of neutrality. It can be seen that for *esc32f* for instance it is quite a bit lower, but mostly it is around 0.62. The "peak" features (ic* and dbi*) do not introduce additional insights as these correlate almost entirely with dbi.

Correlations among those features are shown in Table 6 using Pearson's R. The upper diagonal shows the significance of the correlation using a Bonferroni adjustment to account for multiple comparisons[3]. There are four feature groups that can be identified from these correlations: (1) ac1 and corrlen, (2) ic, pic, $H(x)$, (3) dbi, ic*, dbi*, (4) reg. and div. It is thus a relevant question to which precision a certain feature needs to be measured in order to rely on its value. Is a random walk of, e.g. length 10,000 enough?

[3]Computation has been performed using the R statistical software and the `corr.test` function from the `psych` package

Table 5: Fitness Landscape values for randomly chosen instances of problem instances from various QAP libraries reduced to size 30.

| | ac1 | corrlen | ic | dbi | pic | ic* | dbi* | reg. | div. | $H(X)$ |
|---|---|---|---|---|---|---|---|---|---|---|
| dre110-30 | 0.873 | 40 | 0.779 | 0.637 | 0.425 | 0.825 | 0.652 | 0.001 | 0.002 | 0.954 |
| dre30 | 0.867 | 42 | 0.434 | 0.621 | 0.518 | 0.827 | 0.650 | 0.002 | 0.003 | 0.664 |
| dre42-30 | 0.868 | 70 | 0.456 | 0.620 | 0.515 | 0.828 | 0.648 | 0.002 | 0.003 | 0.682 |
| RAND-S...ci-30 | 0.885 | 70 | 0.391 | 0.624 | 0.518 | 0.825 | 0.652 | 0.928 | 0.966 | 0.631 |
| RAND-S...bl-30 | 0.868 | 33 | 0.817 | 0.648 | 0.390 | 0.817 | 0.648 | 0.000 | 0.000 | 0.990 |
| RAND-S...bl-30 | 0.870 | 36 | 0.825 | 0.646 | 0.336 | 0.825 | 0.646 | 0.000 | 0.000 | 0.996 |
| RAND-S...ci-30 | 0.890 | 76 | 0.391 | 0.624 | 0.519 | 0.826 | 0.651 | 0.948 | 0.973 | 0.631 |
| RAND-S...ci-30 | 0.890 | 56 | 0.391 | 0.624 | 0.519 | 0.825 | 0.651 | 0.935 | 0.968 | 0.631 |
| RAND-S...bl-30 | 0.871 | 35 | 0.707 | 0.623 | 0.462 | 0.825 | 0.650 | 0.000 | 0.000 | 0.888 |
| RAND-S...ci-30 | 0.877 | 67 | 0.391 | 0.623 | 0.520 | 0.827 | 0.649 | 0.929 | 0.963 | 0.631 |
| esc32a-30 | 0.868 | 55 | 0.733 | 0.628 | 0.451 | 0.827 | 0.649 | 0.000 | 0.000 | 0.912 |
| esc32c-30 | 0.875 | 70 | 0.826 | 0.652 | 0.353 | 0.826 | 0.652 | 0.000 | 0.000 | 1.000 |
| esc32d-30 | 0.871 | 37 | 0.827 | 0.650 | 0.362 | 0.827 | 0.650 | 0.000 | 0.000 | 0.999 |
| esc32e-30 | 0.866 | 38 | 0.723 | 0.551 | 0.225 | 0.723 | 0.551 | 0.000 | 0.000 | 0.865 |
| esc32f-30 | 0.866 | 38 | 0.723 | 0.551 | 0.225 | 0.723 | 0.551 | 0.000 | 0.000 | 0.865 |
| esc32h-30 | 0.872 | 41 | 0.820 | 0.647 | 0.386 | 0.828 | 0.647 | 0.000 | 0.000 | 0.992 |
| esc64a-30 | 0.876 | 56 | 0.740 | 0.567 | 0.236 | 0.740 | 0.567 | 0.000 | 0.000 | 0.887 |
| kra32-30 | 0.887 | 62 | 0.534 | 0.617 | 0.502 | 0.826 | 0.651 | 0.007 | 0.011 | 0.744 |
| lipa30b | 0.869 | 33 | 0.394 | 0.623 | 0.522 | 0.826 | 0.650 | 0.027 | 0.049 | 0.633 |
| lipa40b-30 | 0.875 | 76 | 0.393 | 0.623 | 0.520 | 0.827 | 0.648 | 0.046 | 0.082 | 0.632 |
| lipa50a-30 | 0.871 | 60 | 0.482 | 0.618 | 0.512 | 0.828 | 0.648 | 0.001 | 0.002 | 0.702 |
| lipa60b-30 | 0.872 | 95 | 0.392 | 0.623 | 0.521 | 0.827 | 0.648 | 0.092 | 0.161 | 0.631 |
| lipa90b-30 | 0.872 | 49 | 0.392 | 0.623 | 0.521 | 0.828 | 0.647 | 0.171 | 0.290 | 0.631 |
| sko100c-30 | 0.884 | 56 | 0.444 | 0.622 | 0.512 | 0.826 | 0.652 | 0.003 | 0.005 | 0.673 |
| sko100d-30 | 0.883 | 52 | 0.442 | 0.622 | 0.513 | 0.824 | 0.654 | 0.003 | 0.005 | 0.672 |
| sko100e-30 | 0.884 | 62 | 0.441 | 0.622 | 0.513 | 0.825 | 0.653 | 0.003 | 0.005 | 0.670 |
| sko100f-30 | 0.881 | 60 | 0.442 | 0.622 | 0.513 | 0.826 | 0.653 | 0.003 | 0.005 | 0.672 |
| sko42-30 | 0.879 | 59 | 0.457 | 0.620 | 0.513 | 0.827 | 0.650 | 0.002 | 0.004 | 0.683 |
| sko49-30 | 0.878 | 48 | 0.453 | 0.621 | 0.513 | 0.827 | 0.651 | 0.003 | 0.004 | 0.680 |
| sko64-30 | 0.881 | 48 | 0.443 | 0.621 | 0.513 | 0.826 | 0.653 | 0.003 | 0.005 | 0.672 |
| ste36a-30 | 0.895 | 61 | 0.421 | 0.624 | 0.512 | 0.824 | 0.654 | 0.019 | 0.022 | 0.656 |
| ste36b-30 | 0.901 | 61 | 0.401 | 0.626 | 0.512 | 0.824 | 0.655 | 0.077 | 0.125 | 0.640 |
| tai100a-30 | 0.873 | 35 | 0.392 | 0.623 | 0.522 | 0.827 | 0.648 | 0.157 | 0.269 | 0.631 |
| tai30b | 0.869 | 37 | 0.390 | 0.626 | 0.514 | 0.823 | 0.656 | 0.997 | 0.998 | 0.631 |
| tai35a-30 | 0.868 | 41 | 0.392 | 0.623 | 0.521 | 0.828 | 0.648 | 0.168 | 0.282 | 0.631 |
| tai35b-30 | 0.879 | 50 | 0.390 | 0.625 | 0.515 | 0.825 | 0.653 | 0.990 | 0.996 | 0.631 |
| tai40b-30 | 0.872 | 55 | 0.390 | 0.625 | 0.515 | 0.825 | 0.652 | 0.995 | 0.997 | 0.631 |
| tai64c-30 | 0.877 | 68 | 0.619 | 0.440 | 0.164 | 0.619 | 0.440 | 0.116 | 0.036 | 0.725 |
| tai80a-30 | 0.873 | 44 | 0.392 | 0.623 | 0.521 | 0.827 | 0.649 | 0.161 | 0.275 | 0.631 |
| tai80b-30 | 0.888 | 69 | 0.390 | 0.625 | 0.516 | 0.825 | 0.652 | 0.984 | 0.994 | 0.631 |
| tho30 | 0.880 | 58 | 0.394 | 0.624 | 0.518 | 0.825 | 0.652 | 0.050 | 0.080 | 0.634 |
| tho40-30 | 0.885 | 68 | 0.395 | 0.623 | 0.520 | 0.825 | 0.652 | 0.042 | 0.072 | 0.634 |
| wil100-30 | 0.884 | 79 | 0.440 | 0.622 | 0.513 | 0.826 | 0.651 | 0.004 | 0.006 | 0.670 |
| tai45e01-30 | 0.865 | 39 | 0.445 | 0.621 | 0.513 | 0.827 | 0.649 | 0.067 | 0.097 | 0.674 |
| tai45e02-30 | 0.868 | 44 | 0.456 | 0.621 | 0.512 | 0.826 | 0.651 | 0.082 | 0.123 | 0.682 |
| tai45e04-30 | 0.869 | 34 | 0.445 | 0.622 | 0.513 | 0.826 | 0.650 | 0.084 | 0.124 | 0.674 |
| tai45e05-30 | 0.867 | 38 | 0.430 | 0.623 | 0.515 | 0.826 | 0.651 | 0.087 | 0.131 | 0.662 |
| tai45e08-30 | 0.864 | 30 | 0.447 | 0.621 | 0.514 | 0.826 | 0.650 | 0.065 | 0.094 | 0.675 |
| tai45e09-30 | 0.869 | 32 | 0.439 | 0.622 | 0.514 | 0.825 | 0.652 | 0.082 | 0.122 | 0.669 |
| tai45e10-30 | 0.869 | 41 | 0.450 | 0.621 | 0.512 | 0.825 | 0.651 | 0.081 | 0.117 | 0.678 |
| Coeff. Var. | 0.010 | 0.289 | 0.302 | 0.051 | 0.193 | 0.045 | 0.058 | 1.833 | 1.671 | 0.172 |

Table 6: Pearson's correlation coefficient ($R$) among features obtained from a random walk as given in Table 5

| | ac1 | corrlen | ic | dbi | pic | ic* | dbi* | reg. | div. | $H(x)$ |
|---|---|---|---|---|---|---|---|---|---|---|
| ac1 | 1.00 | ** | | | | | | | | |
| corrlen | 0.57 | 1.00 | | | | | | | | |
| ic | -0.32 | -0.28 | 1.00 | | *** | | | | | *** |
| dbi | 0.05 | -0.09 | -0.09 | 1.00 | ** | *** | *** | | | |
| pic | 0.24 | 0.14 | -0.78 | 0.64 | 1.00 | *** | *** | | | *** |
| ic* | 0.10 | -0.02 | -0.33 | 0.96 | 0.82 | 1.00 | *** | | | |
| dbi* | 0.14 | -0.03 | -0.33 | 0.96 | 0.82 | 1.00 | 1.00 | | | |
| reg. | 0.26 | 0.21 | -0.39 | 0.07 | 0.25 | 0.11 | 0.12 | 1.00 | *** | |
| div. | 0.25 | 0.19 | -0.43 | 0.11 | 0.29 | 0.15 | 0.16 | 0.99 | 1.00 | |
| $H(X)$ | -0.31 | -0.30 | 0.99 | 0.04 | -0.70 | -0.21 | -0.21 | -0.38 | -0.42 | 1.00 |

We will thus perform additional experiments that perform 100 repetitions on a set of instances with a maximum length of $2^{18}$ and then subsample this by considering the first $2^i$ values only with $i \in \{7, 8, ..., 18\}$.

In Figure 22 a comparison is given among several well-known features collected from walks of different sizes. It is visible that some feature distributions change notably as the walk length increases. We can hypothesize of two potential reasons for such an observation: On the one hand, it may be that the feature is biased and that the difference in distributions is not caused by a difference in the landscape, but on the other hand it may be that the landscape at a very local level has different properties than on a global level. We do not aim to further the study with respect to isotropy, but refer to Pitzer who studied this in more detail [Pit13].

### 3.3.2 Problem Instance Identification

The motivation for performing landscape analysis is to identify similar problem instances. But many landscape analysis techniques are stochastic as we have shown. In this section we analyze how well we are able to *reidentify* a problem instance that we have already observed. Thus we design a two-phased test in which the exploratory landscape procedures are applied on a set $P$ of $n$ problem instances once in each phase. The first time we obtain our training data, i.e. the features that we would put in the knowledge base and which are assumed our true characteristics [BPWA17]. The second time, we obtain our test data, i.e. the features that we would then compare to the knowledge base.

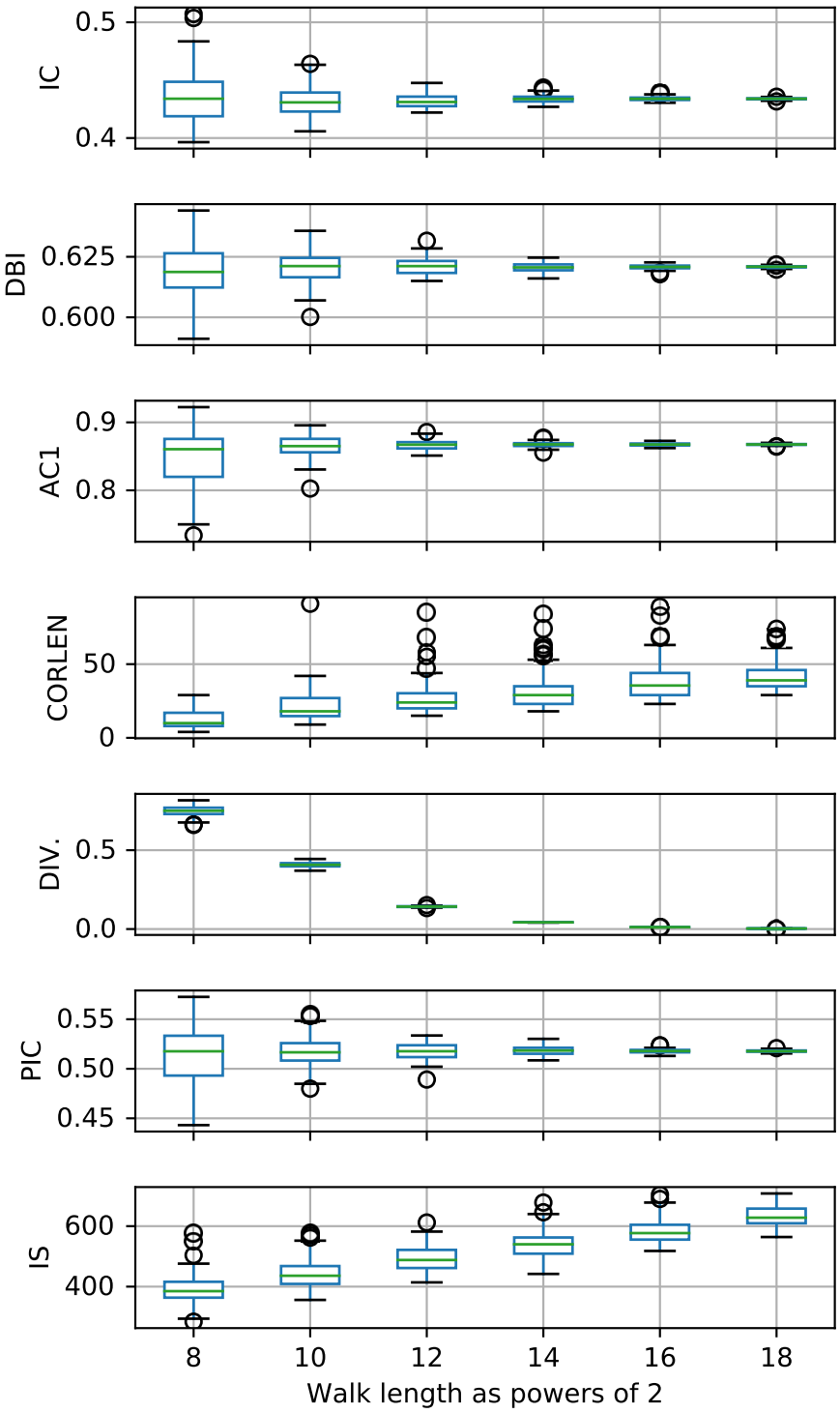



Figure 22: Feature distribution of random walks of various lengths (powers of 2) for the *dre30* instance.

In this evaluation, however we posses the additional information that there is an identical instance in the training and test set. The features of this instance should thus be very similar, if not identical in both data sets. The evaluation is thus simple: For each feature vector $p$ corresponding to one problem instance in our test data we compute the Euclidean distance to all feature vectors in the training data. We then obtain a ranking by sorting those distances in ascending order and obtain the rank $r_p$ of the identical twin of $p$. If the rank equals 1 we have correctly reidentified the problem instance, but the higher the rank the worse. A rank of $n$ would be the worst possible case, but a baseline can be established in form of a random ranking where $E[r_P] = \frac{n}{2}$.

Part of the results were already published by Beham et al [BPWA17]. Here I present an extended study taking into account additional features, varying the problem dimensionality and including more of the walks described above. Similar to the earlier study [BPWA17] the set of problem instances are sampled randomly from various libraries, i.e. QAPLIB [BKR97], Drezner [DHTT05], Microarray [dR06], and Taillard[4]. From Taillard's library only 10 instances are chosen in order to preserve diversity in the set. A downsampling is applied similar to Beham et al. [BPWA17] where facilities and locations are randomly discarded from the weights and distances matrix respectively. Thus, we can create problem instances of the desired size.

The experimental procedure then includes to perform a number of walks on each of these instances, calculate features and create two independent knowledge bases. The first knowledge base is called "training-kb" and the second is called "test-kb". Each consist of the same amount of problem instances, and associated features obtained from mutually exclusive sets of walks. The assumption is that the two knowledge bases hold the same information on each problem instance. However, due to the stochasticity of the exploratory method this may not always be the case, especially short walks exhibit a larger distribution of the feature space as shown in Figure 22. Next, the instances described in the "test-kb" are matched to those present in the "training-kb" as explained above according to a distance in the described feature vector. This process is shown in Figure 23. In the following, our sample consists of 20 problem instances and the sampling is repeated 10 times. Each time a completely new set of problem instances is sampled and the directed walks are applied to it. We apply the method to problem instances of sizes $d \in \{20, 30, 40\}$ separately and average the results and compare different feature sets.

[4] `http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html`

- **SBF** consist of only the features sharpness, bumpiness, flatness as have been described in Equations (3.6) to (3.8).
- **RUG** consist of features associated with ruggedness $ac1$ and $corlen$.
- **IAL** consist of features from the information analysis $ic$, $dbi$, $pic$, $reg.$, $H(X)$, $ic*$, $dbi*$. It is a combination of IAL_REG which uses the formula as given by [VFM00], while IAL_SYM uses the adjustements with respect to symmetry as introduced and analyzed in Section 3.2.2.

As problem instances are rarely of the same size, thus we have to tailor the problem instance set. Instances smaller than the observed dimension are discarded, while larger problem instances are downsampled in an unbiased manner [BPWA17].

Table 7 and Table 8 are looking very similar, but measure the accuracy on two different levels. Table 7 shows results from determining the correct *problem class*. As for the definition of such a class, we use the authors / instance generators that created the problem instances. It is assumed that these generators should significantly differ from one another, but that multiple instances of the same class may not be discernible that easily as they are very similar. Since also, algorithm performance is expected to be rather similar when a problem instance of the same class respectively generator is observed, a high accuracy here should be achieved.

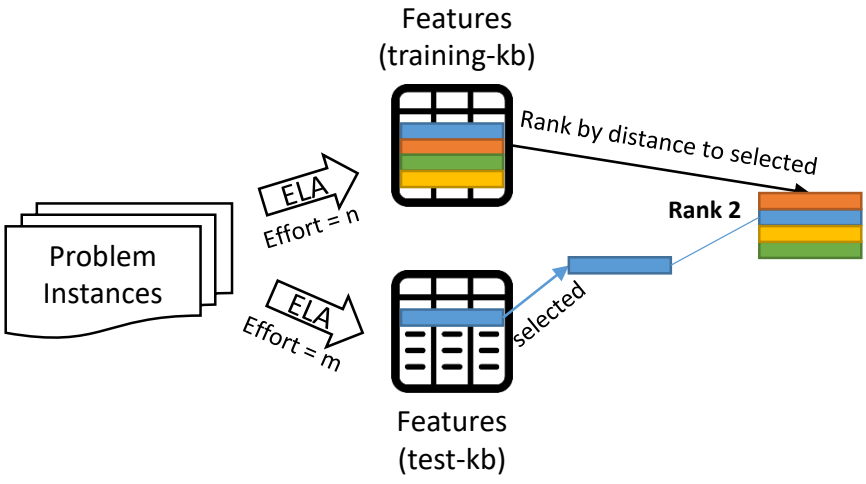



Figure 23: The process of reidentifying problem instances using the stochastic landscape characteristics is shown graphically. In this figure only a single ranking is obtained from a single selected problem instance.

Results in Table 7 show that directed walks are well suited to identify the right problem class. After 500 paths have been observed, features may correctly identify the right problem class in nearly all of the cases using various sets of features. Only the ruggedness-based features are performing not as good indicating that autocorrelation and correlation length of these paths are not very good features. The introduced SBF features perform slightly worse compared to those from information analysis (IAL). A combination of SBF and IALSYM works best for identifying the correct problem class.

In Table 8 results are shown for the case when the exact same problem instance needs to be identified. The data shows again that the *(ll)-dw* and *(li)-dw* variant yield the lowest ranks for the feature sets SBF, IAL and especially the combination SBF and IAL. The problem library consists of on average 100 instances over all dimensions. The task of reidentifying the exact same problem instance is however considerably more difficult, because instances from the same class should be rather similar to each other. In comparison the *(rr)-dw* variant needs about 4 times more paths to achieve e.g. an average rank of 4. Nevertheless, computationally this is still cheaper as inverse directed walks are also the most expensive variant. However, considering that such landscape analysis be integrated into algorithms, we may observe not as many paths and, this is the most important point, the cost of achieving a local optimum is part of the search and not part of the landscape analysis. Here, this cost is added to the analysis method. But, apart from e.g. evolutionary algorithms, many other algorithms consider local optima as important points in the search and strive to improve the search by finding better local optima. Thus, this part of the search cost can be shared with the heuristic approach. In practice both approaches might benefit, the search algorithm may identify the problem instance in comparison to previously observed instances and the landscape analysis might even find new better solutions, for instance using the *(li)-dw* variant. Table 9 compares the effort of the different directed walk types. Due to the added computational effort in determining local optima *(li, ll, rl)-dw* require significantly more time than simple *(rr)-dw*. In addition in inverse directed walks the neighborhood consists of many more members and takes longer to evaluate.

In comparison we also applied random walks to problem instance identification. In Table 10 we observe that random walks are also suited to identify the correct problem class and problem instance. However, it is visible that the ruggedness features are rather misleading. The IAL feature group alone

Table 7: Average rank in identifying problem instance *classes* with various types of directed walks using a first-improvement neighborhood selection strategy (lower is better). The column headers denote the amount of paths that have been performed.

| Paths | 1 | 2 | 5 | 10 | 20 | 50 | 100 | 200 | 500 |
|---|---|---|---|---|---|---|---|---|---|
| (rr)-dw | | | | | | | | | |
| RUG | 11.6 | 13.1 | 10.2 | 9.2 | 8.1 | 6.0 | 6.0 | 4.8 | 4.0 |
| SBF | 8.5 | 7.3 | 5.3 | 4.4 | 3.3 | 2.2 | 1.9 | 1.5 | 1.2 |
| IALREG | 8.5 | 6.1 | 4.3 | 3.4 | 2.3 | 1.8 | 1.5 | 1.4 | 1.2 |
| IALSYM | 8.4 | 7.3 | 4.4 | 3.5 | 2.4 | 1.8 | 1.5 | 1.3 | 1.2 |
| IAL | 9.1 | 7.0 | 4.7 | 3.5 | 2.3 | 1.9 | 1.6 | 1.4 | 1.2 |
| SBF_RUG | 7.8 | 8.1 | 5.1 | 4.1 | 3.4 | 2.4 | 2.4 | 2.2 | 1.3 |
| RUG_IAL | 8.7 | 7.5 | 4.5 | 3.1 | 2.4 | 2.0 | 1.8 | 1.6 | 1.2 |
| SBF_IAL | 8.1 | 5.7 | 3.8 | 2.9 | 2.0 | 1.5 | 1.4 | 1.2 | 1.1 |
| SBF_IALREG | 7.5 | 5.1 | 3.5 | 2.7 | 1.9 | 1.4 | 1.3 | 1.2 | 1.1 |
| SBF_IALSYM | 7.5 | 5.6 | 3.6 | 2.6 | 1.9 | 1.5 | 1.3 | 1.2 | 1.1 |
| (rl)-dw | | | | | | | | | |
| RUG | 12.4 | 10.0 | 10.6 | 7.4 | 6.4 | 5.8 | 4.8 | 4.9 | 3.6 |
| SBF | 7.9 | 7.0 | 4.8 | 4.6 | 3.5 | 2.3 | 1.8 | 1.6 | 1.3 |
| IALREG | 6.8 | 5.8 | 3.9 | 2.8 | 2.2 | 1.7 | 1.4 | 1.1 | 1.1 |
| IALSYM | 7.0 | 6.0 | 3.9 | 3.0 | 2.1 | 1.7 | 1.4 | 1.2 | 1.1 |
| IAL | 7.2 | 6.1 | 4.1 | 3.0 | 2.2 | 1.7 | 1.4 | 1.2 | 1.1 |
| SBF_RUG | 8.5 | 6.4 | 5.2 | 4.0 | 3.1 | 2.0 | 2.3 | 2.2 | 1.7 |
| RUG_IAL | 7.3 | 6.1 | 4.1 | 3.0 | 2.1 | 1.6 | 1.6 | 1.4 | 1.2 |
| SBF_IAL | 6.5 | 5.8 | 3.8 | 2.7 | 2.0 | 1.5 | 1.3 | 1.1 | 1.1 |
| SBF_IALREG | 6.1 | 5.6 | 3.6 | 2.4 | 1.9 | 1.4 | 1.3 | 1.1 | 1.1 |
| SBF_IALSYM | 6.4 | 5.6 | 3.6 | 2.5 | 1.8 | 1.4 | 1.2 | 1.1 | 1.0 |
| (ll)-dw | | | | | | | | | |
| RUG | 12.0 | 10.4 | 7.8 | 5.9 | 4.5 | 4.3 | 5.4 | 3.8 | 4.3 |
| SBF | 9.5 | 6.4 | 5.9 | 5.1 | 4.3 | 3.0 | 2.3 | 1.9 | 1.4 |
| IALREG | 6.4 | 5.0 | 2.8 | 1.9 | 1.4 | 1.2 | 1.2 | 1.0 | 1.1 |
| IALSYM | 6.5 | 5.3 | 2.8 | 1.8 | 1.5 | 1.3 | 1.2 | 1.1 | 1.1 |
| IAL | 6.9 | 5.8 | 2.9 | 1.8 | 1.5 | 1.3 | 1.2 | 1.1 | 1.0 |
| SBF_RUG | 9.7 | 6.6 | 4.5 | 3.7 | 2.8 | 2.2 | 2.7 | 2.0 | 2.6 |
| RUG_IAL | 6.9 | 6.0 | 2.7 | 1.8 | 1.4 | 1.3 | 1.4 | 1.2 | 1.2 |
| SBF_IAL | 9.2 | 5.2 | 2.7 | 1.7 | 1.4 | 1.2 | 1.2 | 1.1 | 1.0 |
| SBF_IALREG | 8.9 | 4.4 | 2.6 | 1.7 | 1.3 | 1.2 | 1.1 | 1.0 | 1.0 |
| SBF_IALSYM | 7.9 | 5.0 | 2.7 | 1.6 | 1.4 | 1.2 | 1.2 | 1.0 | 1.0 |
| (li)-dw | | | | | | | | | |
| RUG | 11.9 | 10.6 | 9.3 | 8.0 | 6.4 | 5.1 | 4.0 | 3.8 | 2.9 |
| SBF | 4.7 | 3.6 | 2.9 | 2.3 | 2.2 | 1.7 | 1.5 | 1.4 | 1.1 |
| IALREG | 4.9 | 3.3 | 2.1 | 1.7 | 1.3 | 1.2 | 1.3 | 1.1 | 1.0 |
| IALSYM | 3.8 | 3.3 | 1.9 | 1.7 | 1.3 | 1.2 | 1.3 | 1.1 | 1.1 |
| IAL | 5.0 | 3.1 | 2.1 | 1.7 | 1.3 | 1.2 | 1.3 | 1.1 | 1.0 |
| SBF_RUG | 6.3 | 5.0 | 3.7 | 3.6 | 2.4 | 2.0 | 1.7 | 1.5 | 1.4 |
| RUG_IAL | 4.9 | 3.5 | 2.2 | 1.8 | 1.4 | 1.3 | 1.4 | 1.1 | 1.1 |
| SBF_IAL | 4.3 | 2.5 | 1.7 | 1.5 | 1.3 | 1.2 | 1.2 | 1.1 | 1.0 |
| SBF_IALREG | 4.2 | 2.4 | 1.7 | 1.5 | 1.3 | 1.2 | 1.2 | 1.1 | 1.0 |
| SBF_IALSYM | 3.3 | 2.4 | 1.6 | 1.4 | 1.2 | 1.1 | 1.3 | 1.1 | 1.0 |

Table 8: Average rank in identifying problem instances *exactly* with various types of directed walks using a first-improvement neighborhood selection strategy (lower is better). The column headers denote the amount of paths that have been performed.

| Paths | 1 | 2 | 5 | 10 | 20 | 50 | 100 | 200 | 500 |
|---|---|---|---|---|---|---|---|---|---|
| (rr)-dw | | | | | | | | | |
| RUG | 45.5 | 44.1 | 39.1 | 35.9 | 32.7 | 24.5 | 26.6 | 22.0 | 13.9 |
| SBF | 32.3 | 27.4 | 19.4 | 14.4 | 11.4 | 7.4 | 5.2 | 4.2 | 3.0 |
| IALREG | 35.0 | 25.9 | 20.2 | 14.1 | 9.0 | 7.1 | 5.2 | 3.8 | 2.6 |
| IALSYM | 34.8 | 28.1 | 22.4 | 15.7 | 9.9 | 7.6 | 5.5 | 3.9 | 2.8 |
| IAL | 36.3 | 28.2 | 23.1 | 16.5 | 9.9 | 7.6 | 5.5 | 4.0 | 2.7 |
| SBF_RUG | 35.1 | 31.6 | 22.3 | 17.1 | 14.8 | 10.8 | 13.8 | 11.8 | 5.2 |
| RUG_IAL | 36.0 | 27.8 | 22.4 | 16.0 | 10.5 | 9.1 | 8.1 | 7.4 | 3.6 |
| SBF_IAL | 33.7 | 26.0 | 20.0 | 13.3 | 7.7 | 5.9 | 4.6 | 3.5 | 2.5 |
| SBF_IALREG | 31.6 | 23.3 | 16.5 | 10.6 | 6.9 | 5.3 | 4.2 | 3.2 | 2.2 |
| SBF_IALSYM | 32.2 | 25.8 | 18.7 | 12.2 | 7.2 | 5.7 | 4.4 | 3.4 | 2.4 |
| (rl)-dw | | | | | | | | | |
| RUG | 43.6 | 39.5 | 43.3 | 30.0 | 24.1 | 18.9 | 19.1 | 19.0 | 18.3 |
| SBF | 31.4 | 24.3 | 18.5 | 14.1 | 10.9 | 6.6 | 4.5 | 3.6 | 2.5 |
| IALREG | 30.8 | 24.6 | 18.0 | 13.1 | 8.4 | 5.1 | 4.0 | 3.1 | 2.4 |
| IALSYM | 28.6 | 26.5 | 19.8 | 14.5 | 8.9 | 5.3 | 4.2 | 3.3 | 2.5 |
| IAL | 30.6 | 26.2 | 20.1 | 14.9 | 9.4 | 5.3 | 4.2 | 3.2 | 2.6 |
| SBF_RUG | 33.4 | 25.5 | 21.5 | 13.7 | 10.6 | 7.3 | 9.8 | 9.8 | 8.7 |
| RUG_IAL | 31.1 | 26.0 | 20.3 | 14.4 | 8.9 | 5.6 | 6.3 | 5.3 | 4.1 |
| SBF_IAL | 28.5 | 22.7 | 16.9 | 11.8 | 7.2 | 4.3 | 3.5 | 2.7 | 2.2 |
| SBF_IALREG | 28.2 | 20.6 | 14.3 | 9.7 | 6.1 | 3.9 | 3.2 | 2.5 | 2.0 |
| SBF_IALSYM | 26.5 | 22.3 | 16.2 | 11.1 | 6.7 | 4.1 | 3.3 | 2.8 | 2.1 |
| (ll)-dw | | | | | | | | | |
| RUG | 42.1 | 37.3 | 31.3 | 25.6 | 19.2 | 13.9 | 22.1 | 20.6 | 20.9 |
| SBF | 43.9 | 30.2 | 21.6 | 17.8 | 14.9 | 9.7 | 6.2 | 4.4 | 2.6 |
| IALREG | 31.6 | 21.5 | 12.5 | 8.0 | 5.6 | 3.7 | 2.9 | 2.2 | 1.6 |
| IALSYM | 31.7 | 24.0 | 13.5 | 8.5 | 6.1 | 3.9 | 3.2 | 2.5 | 1.8 |
| IAL | 32.6 | 23.9 | 13.8 | 8.7 | 5.9 | 3.8 | 3.0 | 2.4 | 1.6 |
| SBF_RUG | 41.4 | 29.2 | 22.0 | 16.3 | 12.9 | 8.8 | 13.5 | 12.1 | 11.6 |
| RUG_IAL | 31.6 | 23.6 | 14.0 | 8.6 | 6.8 | 4.4 | 5.3 | 4.9 | 3.9 |
| SBF_IAL | 40.6 | 22.3 | 13.1 | 8.1 | 5.8 | 3.7 | 2.9 | 2.2 | 1.5 |
| SBF_IALREG | 38.9 | 20.1 | 11.9 | 7.8 | 5.6 | 3.6 | 2.7 | 2.0 | 1.4 |
| SBF_IALSYM | 36.4 | 22.5 | 12.8 | 7.9 | 5.8 | 3.8 | 2.9 | 2.2 | 1.6 |
| (li)-dw | | | | | | | | | |
| RUG | 41.7 | 42.6 | 32.7 | 25.9 | 24.1 | 18.8 | 15.9 | 14.4 | 14.8 |
| SBF | 22.6 | 16.4 | 10.5 | 8.0 | 5.7 | 4.3 | 3.4 | 2.5 | 1.8 |
| IALREG | 20.8 | 15.5 | 7.8 | 5.0 | 4.7 | 3.5 | 2.8 | 2.3 | 1.8 |
| IALSYM | 19.0 | 14.1 | 6.6 | 5.0 | 4.6 | 3.4 | 2.7 | 2.3 | 1.8 |
| IAL | 21.2 | 14.8 | 7.4 | 5.0 | 4.6 | 3.4 | 2.6 | 2.0 | 1.6 |
| SBF_RUG | 27.3 | 24.7 | 16.8 | 12.9 | 11.0 | 9.0 | 7.4 | 6.5 | 7.2 |
| RUG_IAL | 21.5 | 16.5 | 8.9 | 5.9 | 5.4 | 4.4 | 3.3 | 2.4 | 4.1 |
| SBF_IAL | 19.1 | 12.7 | 6.2 | 4.4 | 4.3 | 3.1 | 2.3 | 1.8 | 1.5 |
| SBF_IALREG | 18.7 | 13.0 | 6.2 | 4.3 | 4.3 | 3.3 | 2.5 | 1.9 | 1.6 |
| SBF_IALSYM | 17.6 | 11.9 | 5.5 | 4.3 | 4.1 | 3.1 | 2.5 | 2.0 | 1.5 |

Table 9: Average number of evaluated solution equivalents per path for different types of directed walks for problem instances of size 20, 30, and 40 using a best-improvement selection strategy.

| | (rr)-dw | | (rl)-dw | | (ll)-dw | | (li)-dw |
|---|---|---|---|---|---|---|---|
| Evaluations | 63 | | 174 | | 282 | | 1585 |
| Ratio | 1 | : | 2.8 | : | 4.5 | : | 25 |

is much better able to predict the class and also the exact instance than the RUG or the combination of RUG and IAL. The downside of random walks is that they need to be quite long and are thus unlikely to be integrated into metaheuristics. In some ways integrating short random walks may make sense to escape local optima, but, for instance a walk of length $2^{15}$ is computationally wasteful. Thus, random walks will likely be a method for offline analysis.

Table 10: Average problem class and exact instance rank using random walks of various lengths (lower values are better). The row headers $7, 8, 9, ...$ denote the length of the walk in powers of 2.

| Length $2^x$ | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Class Rank | | | | | | | | | | | | |
| RUG | 12.5 | 12.2 | 12.6 | 13.8 | 11.6 | 10.3 | 9.5 | 9.2 | 8.2 | 9.5 | 9.2 | 8.7 |
| IALREG | 5.2 | 3.6 | 2.9 | 2.6 | 2.2 | 1.8 | 1.6 | 1.3 | 1.4 | 1.3 | 1.2 | 1.2 |
| IALSYM | 5.6 | 3.8 | 2.8 | 2.2 | 2.0 | 1.6 | 1.5 | 1.4 | 1.3 | 1.3 | 1.2 | 1.2 |
| IAL | 6.1 | 4.0 | 3.1 | 2.6 | 2.3 | 1.9 | 1.6 | 1.4 | 1.3 | 1.3 | 1.2 | 1.2 |
| ALL | 6.7 | 5.0 | 3.6 | 3.5 | 2.7 | 2.5 | 2.2 | 1.9 | 1.7 | 2.3 | 2.9 | 3.3 |
| Exact Rank | | | | | | | | | | | | |
| RUG | 48.1 | 47.2 | 46.6 | 45.1 | 40.0 | 39.2 | 35.6 | 32.4 | 29.8 | 28.1 | 27.2 | 27.3 |
| IALREG | 22.1 | 14.4 | 11.6 | 9.2 | 7.0 | 5.2 | 4.4 | 3.7 | 3.1 | 2.8 | 2.6 | 2.1 |
| IALSYM | 22.8 | 15.0 | 11.8 | 8.8 | 6.7 | 5.2 | 4.4 | 3.7 | 3.2 | 2.8 | 2.6 | 2.1 |
| IAL | 24.7 | 16.9 | 12.8 | 9.5 | 7.4 | 5.8 | 4.6 | 3.8 | 3.3 | 3.0 | 2.6 | 2.2 |
| ALL | 26.6 | 20.6 | 17.9 | 14.9 | 10.6 | 10.7 | 8.9 | 8.3 | 7.0 | 6.7 | 8.0 | 8.8 |

Table 11: Average number of evaluated solution equivalents for different types of random walks for problem instances of size 20, 30, and 40 in comparison to the number of paths for directed walks.

| Random Walk | | Directed Walks | | | |
|---|---|---|---|---|---|
| | | (rr)-dw | (rl)-dw | (ll)-dw | (li)-dw |
| Length $2^x$ | Evaluations | Paths | | | |
| 7 | 18 | 0 | 0 | 0 | 0 |
| 8 | 37 | 1 | 0 | 0 | 0 |
| 9 | 74 | 1 | 0 | 0 | 0 |
| 10 | 148 | 2 | 1 | 1 | 0 |
| 11 | 296 | 5 | 2 | 1 | 0 |
| 12 | 592 | 9 | 3 | 2 | 0 |
| 13 | 1183 | 19 | 7 | 4 | 1 |
| 14 | 2367 | 38 | 14 | 8 | 1 |
| 15 | 4733 | 75 | 27 | 17 | 3 |
| 16 | 9466 | 151 | 54 | 34 | 6 |
| 17 | 18933 | 301 | 109 | 67 | 12 |
| 18 | 37865 | 602 | 218 | 134 | 24 |

## 3.4 Integration into Metaheuristic Algorithms

So far landscape analysis has mostly been viewed as a process that is performed either in advance of problem solving or as an additional task to gain a better understanding. It is still a rather open topic on how landscape analysis could be integrated into algorithms. Motivations for an integration would be to enhance self-adaptive strategies or to provide additional insights instead of "just" a good solution. Such insights could, for instance, help determine if another algorithm was probably more suited to solve this instance given a knowledge base of pas experiments. In this, rather positional and opinionated, part of the thesis, I want to provide some thoughts and arguments which could lead to a bigger discussion on potential research questions in this topic.

Previous approaches in landscape analysis have mostly defined new sampling methods, new ways of analyzing those samples, and theoretical and empirical results on which inferences are reasonable from those analyses. However, the integration into algorithms poses additional challenges, for instance:

1. Trajectory-based methods are often greedy
2. Population-based methods often converge to highly similar solutions
3. Search dynamics may change if ELA results are reintegrated
4. The efficiency of the search may be degraded

For instance, the greediness of sampling typically used in trajectory-based methods makes it rather difficult to observe characteristics of a random walk. Or put in other words, it's highly unlikely to observe a random walk as part of a problem solving algorithm. We may however experience some form of adaptive walks, for instance local search as an extreme example of adaptive walks. On the other hand, local search is rather short and does not extend beyond the local optimum. But, for instance tabu search or simulated annealing may be viewed as performing an adaptive walk on the landscape.

Another mentioned challenge is the diversity of solutions in a population. For instance, to perform directed walks the requirement is that solutions should be rather dissimilar in order to experience a longer walk that crosses a larger part of the search space. If mostly similar individuals are used can we still uncover relevant landscape properties? However, in a greedy replacement or selection strategy as can be commonly observed in population-based algorithms, diversity is not preserved very well. And even when diversity respectively its loss is accounted for, e.g. as in ALPS [Hor06], the variation operators are still applied to mostly similar individuals.

Another challenge is how to integrate a walk into the dynamic search process of a population-based method. For instance, should solutions found during directed walks be used in the next variation phase of an algorithm? It could potentially have a devastating effect by reducing diversity even faster and thus increasing the rate of premature convergence. Even in trajectory-based methods, adding, e.g. directed walks is not a simple task, considering that typically only the current solution is stored. Also, if characteristics are identified, for instance a high ruggedness, how should the strategy of an algorithm be adapted? A general answer is difficult to give and depends on the dynamics of each algorithm (instance).

The efficiency of the search is another concern that is raised by integrating exploratory methods into metaheuristics. The exploratory walks described above may require a multitude of evaluations which could otherwise be used to advance the dynamic search progress. What cost has to be paid in order to achieve which improvement? How can the user of an optimization method benefit from additional landcape information that may be obtained along with the search?

The time has not yet come to provide satisfactory answers to all these questions. Nevertheless, I hope that this thesis gives at least some insights on how to progress with this idea. For instance, inverse directed walks may be useful in memetic algorithms to diversify the population. Inverse directed walks could expand the population within high quality regions of the search space or explore into neighboring regions. Additional exploration could be achieved by performing directed walks towards random solutions. The advantage of embedding such walks is that we may obtain landscape characteristics in addition to good solutions.

## 3.5 Conclusions

In this section we explored a new kind of landscape analysis based on directed walks which have been first introduced by myself and co-authors in [BPWA17] and [BAW17]. These are based on the well-known path relinking heuristic. Several new variants of directed walks have been introduced in this thesis that are connecting different types of points and an inverse variant was introduced which differs significantly from path relinking. In addition, three new features have been introduced based on the data obtained from directed walks together with existing features. A bias was found in existing features with respect to symmetry of up and down walks that has been described and where a remedy is proposed. It has been shown that the effect of that bias was small however. A larger study has been conducted on problem instances of the quadratic assignment problem showing the accuracy of those features as a function of the amount of data obtained. Finally, the integration of directed walks and thus of exploratory landscape into the search dynamics of metaheuristic algorithms has been discussed.

It is shown that directed walks bear some significant costs, potentially even more than random walks, but may also uncover better solutions. In addition, directed walks are able to analyze the landscape at neuralgic points such as local optima which are usually of more relevance to a search algorithm than an unbiased randomly sampled solution.

The main motivation for the usage of directed walks instead of random, adaptive or up/down walks as previously introduced is their usability as part of a dynamic search process. While certainly it may be beneficial to conduct a random walk in order to escape some region of the search space, these are not typically included in search algorithms or are only very short. Using typical search operators to identify landscape feature has not yet been discussed widely and such "landscape-aware" heuristic methods are yet to emerge on a wider scale. The next steps would be to develop and describe such methods that are able to identify good solutions and provide an analysis of the solution space at the same time. In this regard [BPWA17] was a first attempt which showed that there are still some challenges that need to be overcome and which have been discussed in the previous section in more detail.

# 4 Algorithm Selection Case Studies

> "At all levels of organization life depends on the maintenance of a certain balance among its factors."
>
> Sewall Wright [Wri32]

Assignment problems such as the ones that were described in Section 2.1 come in various forms. An assignment may be made to two groups in which case a binary choice is made for each item, assignments may be made such that each item is assigned to exactly one group for which the representation of a solution as a *permutation* is quite efficient. Additionally, assignment may occur such that multiple items are assigned to the same group and not all group may receive items for which a discrete encoding needs to be used. In this section we have chosen the quadratic assignment problem (QAP) which is a representative of the permutation-assignment problem and its generalized extension the GQAP to which solutions are represented in form of a discrete encoding and for which the solution space may contain infeasible solutions.

The case studies presented here describe and analyze features of the presented problems. They consider a certain set of benchmark instances on which the study is performed and a certain set of algorithm instances that are then benchmarked. Those results are analyzed in that landscape properties are related to performance measures and performances among solvers is analyzed. Finally, a combined algorithmic approach is chosen with a recommender as the orchestrator that considers all presented algorithms and decides case-by-case, i.e., for each problem instance, which of these algorithms is to be applied. Finally, the performance of that combined solver is contrasted with the performance of the individual solvers. We will see that this approach can lead to improved results as the combination outperforms individual solvers.

## 4.1 Algorithm Selection for Solving Quadratic Assignment Problems

In this section I present a summary and extension of the work that has been done by myself, Michael Affenzeller, and Stefan Wagner and has been presented at GECCO 2017 [BAW17]. The main contributions of this part to the state of the art are:

1. Benchmarking and comparison of a wide range of different metaheuristics
2. Evaluation of an algorithm instance selector

The work described by Beham et al. [BAW17] was distributed such that Beham devised the methodological approach and scope, devised and implemented the algorithm instances (if not otherwise stated), chose the problem instances used in the study, and carried out the experiments. Beham implemented and carried out the landscape analysis and generated, and analyzed the algorithm selection results. Affenzeller and Wagner both contributed with comments, engaged in discussions, and supported the editing process of the final text.

### 4.1.1 Feature Extraction

A work on algorithm selection, typically starts with a description of the features which will be used to characterize the problem instances [Ric76]. These have to be embedded into some vector space in order to allow relating them to one another. Algorithm instances on the other hand are the inferred or predicted object and thus do not require such an embedding, although we have given some thoughts on some categorizations in Section 2. In this study, we will consider each algorithm instance as a unique entity of which only its performance is known.

Among the characteristics that are used in this study, we will evaluate problem specific features which are computed as statistical measures of the problem data, i.e. the weights and distances matrix of the QAP instances. But also, sampling specific features are used. Among them we chose well-known features from the literature as has been described in Section 2.3, but also new features introduced in Section 3. Problem specific features such as flow dominance have been suggested in the past [Stü06]. The three different problem specific characteristics that are considered are dominance, sparsity, and asymmetry. Given our domain for the permutation $\mathbb{D} = [1; N]$ and square

matrix $A = N \times N$ dominance is defined as the coefficient of variation of the matrix' values $a_{ij}$, see Equation (4.1). Sparsity is the relative number of cells with a value of 0, see Equation (4.2), and asymmetry is the relative number of symmetric pairs that are not equal ($a_{ij} \neq a_{ji}$), see Equation (4.3).

$$\texttt{Dominance:} \quad \frac{\sigma_A}{\mu_A} \tag{4.1}$$

$$\texttt{Sparsity:} \quad \frac{\{(i,j) \in \mathbb{D} \times \mathbb{D} \mid a_{ij} = 0\}}{N \cdot N} \tag{4.2}$$

$$\texttt{Asymmetry:} \quad \frac{2 \cdot \{(i,j) \in \mathbb{D} \times \mathbb{D} \mid i < j \wedge a_{ij} \neq a_{ji}\}}{N \cdot (N-1)} \tag{4.3}$$

In addition we consider features from an exploratory-based approach based on directed walks as described in Section 3.1.

### 4.1.2 Algorithm Instances

In the following we will introduce the algorithms and corresponding instances, some of which have been analyzed in previous studies [BAP15], and which are also applied in this study [BAW17]:

1. Robust Tabu Search (3 instances)
2. Standard Tabu Search (1 instance)
3. Variable Neighborhood Search (1 instance)
4. Iterated Genetic Algorithm (1 instance)
5. Memetic Algorithms (2 instances)
6. Multi-start Local Search (1 instance)
7. Random Search (1 instance)

An algorithm instance is created by choosing the respective parameter values. In this regard we chose to use not only a fixed value parameterization, but to consider the influence of the problem dimension already in the parameterization. For instance, robust tabu search defines an aspiration condition for which it is a priori meaningful to consider longer aspiration conditions for larger problems. We define to scale this parameter with the dimension using a constant factor as weight.

If an algorithm is prone to terminate or converge without exploiting the computational budget, we chose to implement an independent restart strategy.

This is often much less of a problem in practical applications when time is a severely limiting factor, but is common in benchmark scenarios when the convergence behavior is analyzed in more detail. For example, genetic algorithms converge after a certain number of generations and then continue running, but with mutation being the most important contributor to the variation process. At the same time, mutation is applied with low probability, thus the search process beyond convergence is somewhat inefficient. In preliminary experiments on the quadratic assignment problem convergence was observed within a few hundred generations. Thus, the iterated genetic algorithm makes use of independent restarts until the computational budget is exploited. The best quality achieved in any of the restarted runs is considered the algorithm's result. Thus all algorithm instances achieve the same number of evaluated solutions, which is the runtime measure on which they are compared against.

All algorithm instances in this study have been implemented in C# using the .NET Framework 4.5 and the HeuristicLab 3.3.13 optimization environment. Implementing the algorithms in one framework facilitates testing and benchmarking in that there is a common interface to running each algorithm.

**Robust Tabu Search**

Robust tabu search (originally called robust taboo search) (RTS) was introduced by Taillard [Tai91]. It is aimed to avoid the cycling effect of standard tabu search which can be easily observed with very small tabu lists. Due to the deterministic nature of tabu search (exhaustive neighborhood generation and greedy neighbor selection) the tabu search trajectory may return to previously visited solutions in a cyclic way. It can only escape such a cycle when the tabu list is changed in size or some other forms of aspiration or additional memories perturb the trajectory.

A description of the algorithm is given in Algorithm 5. In the implementation used in this study the originally proposed random choice of tabu tenure in an interval is not used, but the strategy published as code on the author's website[5] has been employed. In RTS the tabu tenure is chosen randomly each time a move is made. However, the most important new aspect concerns the introduction of an "alternative aspiration condition" that is aimed to greatly diversify the search as a kind of long-term memory. This aspiration condition prioritizes moves that have not been observed in a long time. In preliminary experiments the distribution parameter of the tabu tenure showed less of an

[5] `http://mistic.heig-vd.ch/taillard/codes.dir/tabou_qap2.c`

effect on the performance and was set maxtenure = 200. However, the alternative aspiration condition *aspfactor*, i.e. the iterations before a move becomes interesting, showed a strong effect on the performance. Thus, we created three instances and included them in the benchmark:

1. *RTS-noasp* where aspfactor = $\infty$
2. *RTS-20D* where aspfactor = 20
3. *RTS-100D* where aspfactor = 100

Under the setting of aspfactor = $\infty$ the aspiration condition does not come into force and the trajectory is akin to an adaptive walk on the fitness landscape. Only the tabu tenure affects the path and contributes to a diversification of the search. We will see later that this parameterization provided good results overall. For larger *dreXX* instances RTS-noasp was the only algorithm instance to achieve the targets.

In general tabu search is a rather low-level heuristic as can be observed in the description of Algorithm 5. The tabu and aspiration conditions are highly complex and the data structure of the tabu "list" also affects the implementation. In addition, as the tabu list holds often the changes rather than the full solutions, it is bound stronger to the defined neighborhood. It is thus more difficult to provide an abstract definition compared to other algorithms. Tabu search, does not make use of variation heuristics, except an initial solution generation and is performant due to an efficient delta computation of a move's fitness. However, an abstract definition can be achieved on the notion of move and neighborhood. Nevertheless, the data structure of the tabu list is related to the move and is more difficult to generalize. In the case of the implemented algorithm, the tabu list has been implemented as a symmetric matrix that records the last iteration at which two cities (indices) have been swapped.

**Standard Tabu Search**

A "standard" tabu search was implemented using a fixed tabu tenure that was set to a factor of three of the problem dimension. A pseudo code description is given in Algorithm 6. A simpler aspiration criterion was defined which allows reverting a move (swapping two cities into places they both occupied already) when the solution's quality would be better compared to the quality recorded at the time the move was added to the tabu list. Thus, an additional list is used that stores the quality that a future move that reverts

**Algorithm 5** Robust Tabu Search for QAP, adapted from [Tai91]

```
1: procedure RTS(↓ dim, ↓ aspfactor, ↓ maxtenure)
2:     asp ← dim · aspfactor
3:     tabulist_ij ← −[dim ∗ (i + 1) + j + 1]        ∀ i, j ∈ [1..dim]
4:     iter ← 0
5:     sol ← Sample()
6:     fit, bestfit ← Evaluate(sol)
7:     while not Terminate() do
8:         aspired ← False
9:         next ← (0, 0)
10:        nextfit ← ∞
11:        for i ∈ [1, 2, ..., dim − 1] do
12:            for j ∈ [i + 1, i + 2, ..., dim] do
13:                q ← fit + DeltaFit(sol, i, j)
14:                a ← tabulist_ij < iter ∧ tabulist_ji < iter
15:                b ← tabulist_ij < iter − asp ∧ tabulist_ji < iter − asp
16:                c ← q < bestfit
17:                if [(b ∨ c) ∧ ¬aspired] ∨ [(b ∨ c) ∧ aspired ∧ q < nextfit]
18:                    ∨ (a ∧ ¬b ∧ ¬c ∧ ¬aspired ∧ q < nextfit) then
19:                    next ← (i, j)
20:                    nextfit ← q
21:                    if b ∨ c then
22:                        aspired ← True
23:                    end if
24:                end if
25:            end for
26:        end for
27:        if next ≠ (0, 0) then
28:            (i, j) ← next
29:            (r, s) ← (Rand(]0; 1]), Rand(]0; 1]))
30:            tabulist_ij ← iter + r^3 · maxtenure
31:            tabulist_ji ← iter + s^3 · maxtenure
32:            Swap(sol, i, j)
33:            fit ← nextfit
34:            bestfit ← Minimum(fit, bestfit)
35:        end if
36:        iter ← iter + 1
37:    end while
38:    return (sol, fit)
39: end procedure
```

this change has to surpass. In addition, the standard aspiration criterion of an improvement to the best found so far is also enabled.

The STS algorithm is mostly deterministic, as is common for tabu search, except for a random tie breaking when two or more of the best moves in the neighborhood would achieve the same fitness. In that case each move is chosen with the same probability. It also includes a somewhat less-standard element in that neutral moves are rejected. In some QAP instances moves may not make any change to the fitness function and would be preferred over a deteriorating move. These plateaus may however be quite large as we have seen in Section 3 and exploration of such plateau is inefficient due to the random movement. Thus, this STS implementation prefers a deterioration over a movement within the plateau. The tie breaking ensures that plateaus are entered in a random location, thus the search moves rather along the edge of the plateau.

**Variable Neighborhood Search**

Variable neighborhood search (VNS) algorithms are a family of local search-based metaheuristics [MH97]. As has been described in Section 2.2.4 VNS algorithms alternate between local search and random perturbation of the solution. In the perturbation phase multiple different or growing neighborhoods are applied. The neighborhood is changed or grows when the algorithm is repeatedly unsuccessful in obtaining better solutions. In our implementation VNS is elitistic and thus bases the search on the best solution found so far. The local search used in this study is based on an exhaustive enumeration of the *swap2* neighborhood. In the perturbation phase $k$-swaps are performed where $k$ is the strategy parameter that is increased by VNS. A maximum of half the problem dimension was chosen.

A pseudocode description is given in Algorithm 7. The algorithm is less complex to describe than tabu search, but `Swap2Localsearch` also bears some complexity that is described in Algorithm 18 in the appendix. Essentially, it performs a best-improvement local search and stops when no improving move in the neighborhood is possible.

**Iterated Genetic Algorithm**

The genetic algorithm (GA) instance was set to use 1-elitism, a population size of 500, 15% mutation probability (*swap2*), partially-matched crossover (PMX) [Fog88], and tournament selection (group size 3). Convergence is often very fast and running a genetic algorithm past the point of convergence is

**Algorithm 6** Standard Tabu Search for QAP

```
 1: procedure STS(↓ dim, ↓ tenurefactor)
 2:     tenure ← dim · tenurefactor
 3:     sol ← Sample()
 4:     tabulist_ij ← if sol[i] = j then 0 else −tenure end
 5:     history ← [ ]
 6:     iter ← 0
 7:     fit, bestfit ← Evaluate(sol)
 8:     while not Terminate() do
 9:         next ← (0, 0)
10:         nextfit ← ∞
11:         count ← 1
12:         for i ∈ [1, 2, ..., dim − 1] do
13:             for j ∈ [i + 1, i + 2, ..., dim] do
14:                 q ← fit + DeltaFit(sol, i, j)
15:                 if q = fit then continue end
16:                 (r, s) ← (sol[j], sol[i])
17:                 istabu ← tabulist_ir > iter − tenure ∨ tabulist_js > iter − tenure
18:                 (b, c) ← (iter − tabulist_ir, iter − tabulist_js)
19:                 aspired ← bestfit > q ∨ history[b] > q ∨ history[c] > q
20:                 if (¬istabu ∨ aspired)
21:                     ∧ [q < nextfit ∨ (q = nextfit ∧ Rand([0; 1[) · count < 1)] then
22:                     if q = nextfit then count ← count + 1
23:                     else if q < nextfit then count ← 1 end
24:                     next ← (i, j)
25:                     nextfit ← q
26:                 end if
27:             end for
28:         end for
29:         if next ≠ (0, 0) then
30:             (i, j) ← next
31:             (r, s) ← (sol[j], sol[i])
32:             tabulist_ir, tabulist_js ← iter
33:             Add(history, Minimum(fit, q))                ▷ Prune to length tenure
34:             Swap(sol, i, j)
35:             fit ← nextfit
36:             bestfit ← Minimum(fit, bestfit)
37:         end if
38:         iter ← iter + 1
39:     end while
40:     return (sol, fit)
41: end procedure
```

**Algorithm 7** Variable Neighborhood Search for QAP

```
1: procedure VNS(↓ dim, ↓ nbhoodfactor)
2:     K ← dim · nbhoodfactor
3:     sol ← Sample()
4:     (iter, k) ← (0, 0)
5:     fit ← bestfit ← Evaluate(sol)
6:     while not Terminate() do
7:         p ← Swapk(sol, k)
8:         (s, sfit) ← Swap2Localsearch(dim, p)
9:         if sfit < fit then
10:            (sol, fit) ← (s, sfit)
11:            k ← 0
12:        else
13:            k ← k + 1  mod K
14:        end if
15:        bestfit ← Minimum(fit, bestfit)
16:        iter ← iter + 1
17:    end while
18:    return (sol, fit)
19: end procedure
```

inefficient. In this study the algorithm runs for a fixed number of 200 generations. Then it is restarted with a fresh population. This so called "iterated" algorithm continues to run until the total number of evaluated solutions has been used up. A pseudo code description is given in Algorithm 8.

### Memetic Algorithm

The memetic algorithm (MA) introduced in [MF97] as "Genetic Local Search (GLS)" and later published in [MF00] builds upon a tighter integration of crossover and local search. A crossover operator typically retains the information when both parents agree on a certain allele, but randomly chooses one in case of a disagreement. The local search phase of GLS is restricted to only those positions where the crossover found the parents in disagreement, thus exploiting only a restricted search space. The instance is configured to use a population size of 100. Per generation 50 crossovers and 20 mutations (reversal of a range within the permutation) are performed. The local search uses the *swap2* neighborhood in the reduced sub-space given by the two parents. A pseudocode description is given in Algorithm 9 and follows [MF97, MF00]. The method `DiversitySelect` keeps the best unique solutions. The implementation is described in the appendix in Algorithm 19.

**Algorithm 8** Iterated Genetic Algorithm for QAP

```
 1: procedure GA(↓ popsize, ↓ mutprob, ↓ maxgen)
 2:     pop ← Sample(popsize)                          ▷ Initialize the population
 3:     best ← BestOf(pop)
 4:     while not Terminate() do
 5:         for gen in 1..maxgen do
 6:             parents ← TournamentSelect(pop)        ▷ 2 · (popsize − 1)
 7:             nextgen ← PMX(parents)                 ▷ Partially-Matched Crossover [Fog88]
 8:             Swap2(nextgen, mutprob)
 9:             nextgen ←+ BestOf(pop)                 ▷ Add elite solution
10:             pop ← nextgen
11:             Update(best, pop)
12:         end for
13:         pop ← Sample(popsize)                      ▷ Reinitialize
14:     end while
15:     return best
16: end procedure
```

### Iterated Memetic Algorithm

Another variant of a general memetic algorithm is used that builds upon the iterated genetic algorithm described above. Instead of a low-probability random mutation within the *swap2* neighborhood, it uses a full local search in that neighborhood. Because this operation is quite expensive a smaller population size of only 50 individuals is used. Elitism is not used and a linear rank selection is used during parent selection. Convergence is detected when worst and best quality are equal in which case the algorithm is restarted with a new population of local optima derived from an equal number of randomly drawn samples from the whole solution space. A pseudo code description is given in Algorithm 10.

### Multi-start Local Search

Multi-start local search uses the aforementioned local search from randomly sampled starting solutions. The repeated application of local search from many different points in the search space is presumed to be effective if there is a high probability that the global optimum is located in a larger basin of the search space or when local optima are scattered in the search space. The algorithm is shown in Algorithm 11.

**Algorithm 9** Genetic Local Search for QAP [MF97]

```
1: procedure GLS(↓ dim, ↓ #crossovers, ↓ #mutations)
2:     pop ← Sample()                                   ▷ Initialize the population
3:     nextgen ← [ ]
4:     for p in pop do
5:         nextgen ←+ Swap2Localsearch(dim, p)          ▷ Add to next generation
6:     end for
7:     (pop, nextgen) ← (nextgen, [ ])
8:     while not Terminate() do
9:         for c in 1..#crossovers do
10:            (p1, p2) ← Select(pop)
11:            (o, info) ← DPX(p1, p2)                  ▷ Diversity preserving crossover [MF97]
12:            nextgen ←+ Swap2Localsearch(dim, o, info)
13:        end for
14:        for m in 1..#mutations do
15:            p ← Select(pop)
16:            o ← Mutate(p)
17:            nextgen ←+ Swap2Localsearch(dim, o)
18:        end for
19:        (pop, nextgen) ← (DiversitySelect(pop, nextgen), [ ])
20:    end while
21:    return pop
22: end procedure
```

**Algorithm 10** Iterated Memetic Algorithm for QAP

```
1: procedure GA+LS(↓ dim, ↓ popsize)
2:     sample ← Sample(popsize)                         ▷ Initialize the population
3:     best ← BestOf(sample)
4:     while not Terminate() do
5:         pop ← Swap2Localsearch(dim, sample)
6:         best ← BestOf(pop)
7:         while not Converged() do
8:             parents ← LinearRankSelect(pop)          ▷ 2 · (popsize)
9:             offspring ← PMX(parents)                 ▷ Partially-Matched Crossover [Fog88]
10:            nextgen ← Swap2Localsearch(dim, offspring)
11:            pop ← nextgen
12:            Update(best, pop)
13:        end while
14:        sample ← Sample(popsize)                     ▷ Resample
15:    end while
16:    return best
17: end procedure
```

**Algorithm 11** Multi-start Local Search for QAP

```
1: procedure MLS(↓ dim)
2:     sol ← Sample()
3:     (bestsol, bestfit) ← Swap2Localsearch(dim, sol)
4:     while not Terminate() do
5:         (sol, fit) ← Swap2Localsearch(dim, Sample())
6:         if fit < bestfit then
7:             bestfit ← fit
8:             bestsol ← sol
9:         end if
10:    end while
11:    return (bestsol, bestfit)
12: end procedure
```

**Random Search**

Random search draws random samples from the solution space with repetition. It remembers the best solution that it has sampled.

### 4.1.3 Benchmark Data Generation

For this study a total of 47 problem instances were used from the QAPLIB [BKR97], microarray instances [dR06], Drezner's [DHTT05], and Taillard's symmetrical and structured QAP instances[6]. The instances used in the study were chosen such as to include two or more that are alike and with similar or a progressing dimensionality in order to simulate prior knowledge in form of a similar instance present in the database [BAW17]. The instances *els19*, *esc32a*, *had20*, *kra32*, and *nug30* did not have a similar counterpart. All chosen problem instances are shown in Table 14 on page 119.

### 4.1.4 Benchmark Performance Analysis

The performance of metaheuristic algorithms can be measured along multiple dimensions as outlined in Section 2.5 on page 47. In this study we consider two measures: (1) the quality of the solution and (2) the runtime of the algorithm. While the first is simply the solution value, respectively the deviation to the best-known solution value, the second performance bears more complexity. When runtime is measured by the elapsed time measured by the so called

[6] `http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html`

"wall clock" a fair comparison would require that the experiment is performed on the same hardware and under the same load. In addition, implementation details would become relevant as the efficiency of the implementation greatly determines the elapsed wall clock time. To conduct the thousands of runs necessary for this study a *heterogeneous* cluster was available for distributed computing of the individual runs. Thus it was decided to use the amount of evaluations as a measure of the runtime. Nevertheless, using evaluations as measure also requires some considerations. Some of the algorithms make use of an efficient delta calculation to determine the fitness which takes less time to evaluate. It was thus decided to use *evaluated solution equivalents* hereafter simply denoted as "evaluations". These are computed by regarding the fitness function as a summation of $N \cdot N$ products. A delta calculation that requires to recompute only 4 of these products [Tai91] is thus equivalent to $4/(N \cdot N)$ solution evaluations, otherwise put, a full neighborhood evaluation (neighborhood size is $(N \cdot N)/2$) is equivalent to 2 full solution evaluations. In the experiment however sometimes more sums need to be recomputed and it is always tracked exactly how many of these operations are performed.

The basis for all comparisons is the convergence graph that needs to be stored in every run. This graph is monotonic and records the timepoints at which an improving solution is available as well as the quality of that solution. Based on this data one can attempt post-hoc analysis using *fixed-budget* and *fixed-target* comparisons as described in Section 2.5 on page 47.

In Figure 24 the performances of all algorithm instances described in Section 4.1.2 can be compared across all benchmark instances for the 1% target. This target means that the solution has to be at most 1% worse than the best-known quality for the respective problem instance. It is visible that among all intances the tabu search variants are dominating for small to medium run-lengths, but that for larger run-lengths the genetic algorithms and variable neighborhood starts to dominate. An overall well-performing configuration is RTS-ASP100D which is "outperformed" after about 200,000 evaluations by the iterated memetic algorithm. However, to conclude that the other algorithms are not useful is wrong based on this picture. For individual problem instances different algorithm instances are effective. In Figure 25 the results show that RTS-noasp and to a lesser degree STS-3D may solve the dre56 problem instance to the 1% target within the given limit of solution evaluations. In comparison in Figure 26 neither RTS-noasp and STS-3D found the 1% target once in 20 runs.

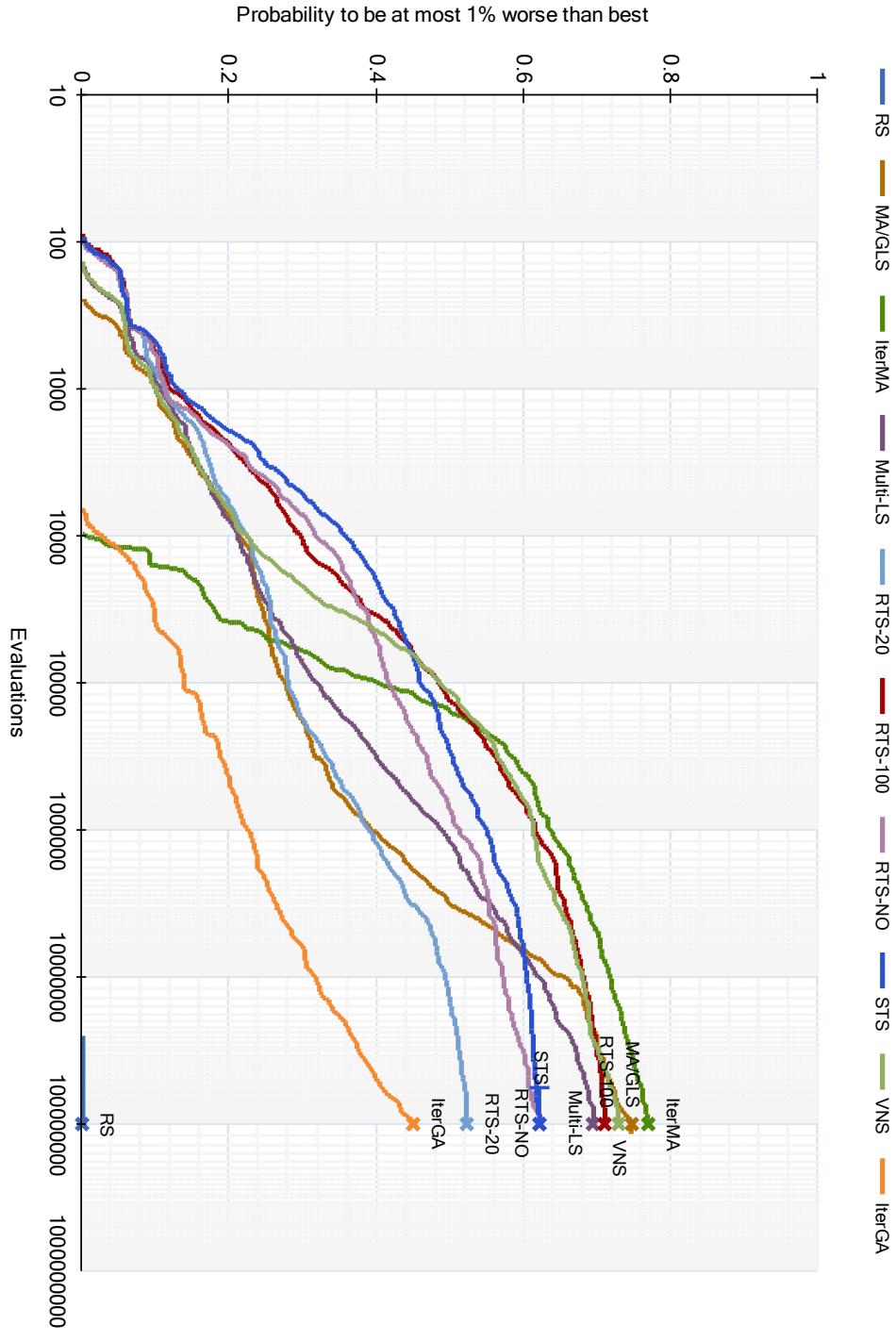



Figure 24: Runtime analysis of different algorithm instances for the 1% target on the benchmark problem set. [BAW17]

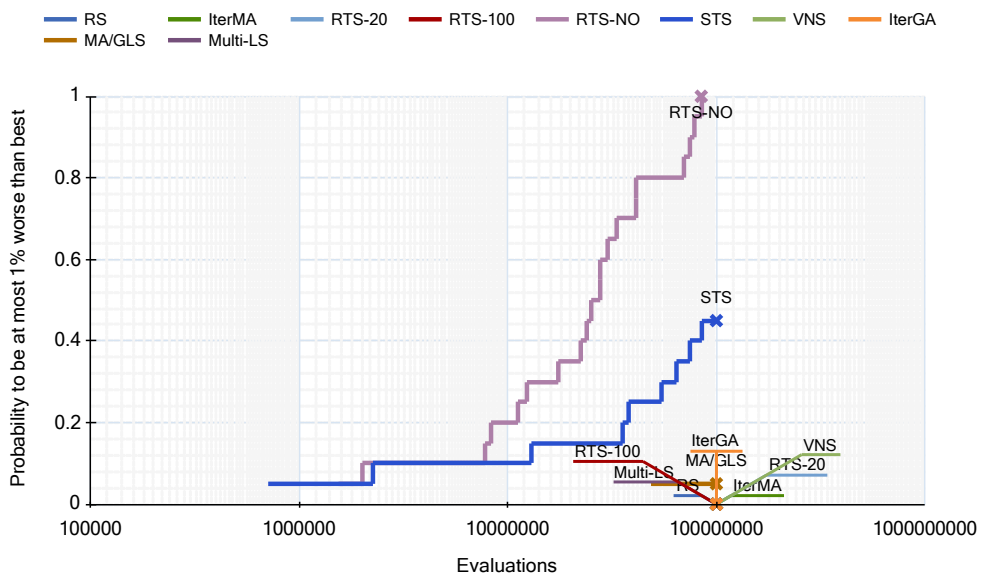



Figure 25: Runtime analysis of different algorithm instances for the 1% target on the dre56 problem instance.

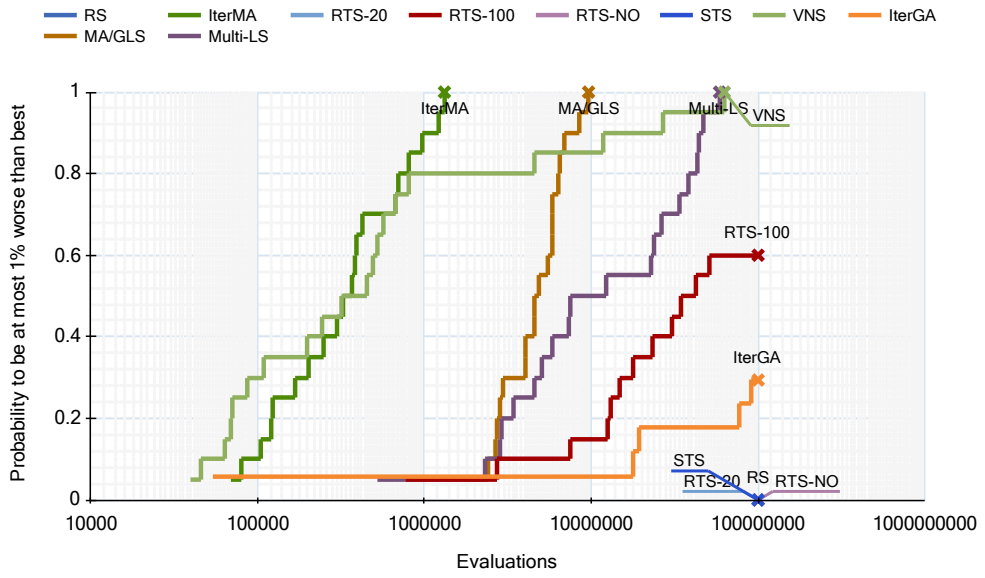



Figure 26: Runtime analysis of different algorithm instances for the 1% target on the tai45e01 problem instance.

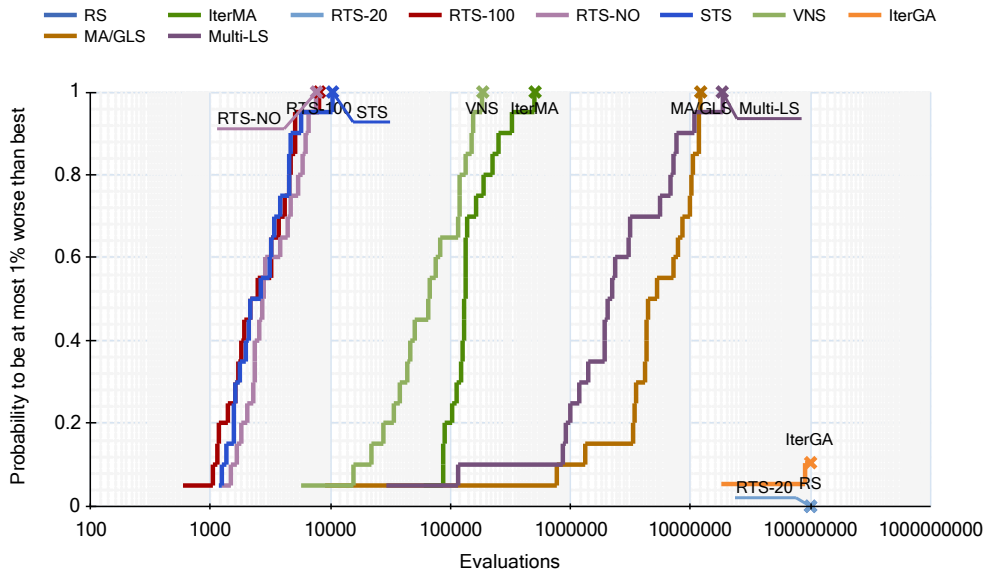



Figure 27: Runtime analysis of different algorithm instances for the 1% target on the lipa50a problem instance.

On some instances, a clustering of the algorithm instances can be observed while others show a more pronounced ranking. For instance, as shown in Figure 27 in lipa50a tabu search variants perform best, followed by VNS and the iterated memetic algorithm, again followed by the memetic algorithm and the multi-start local search. In comparison, on the nug30 instance there is a clear progression of algorithm instances as seen in Figure 28. This has implications when considering algorithm selection. In the case of the lipa50a we would be satisfied to get recommended one of the top three algorithm instances. In the evaluation of the recommendation it was thus decided to group these instances into performance classes as will be detailed later.

In Figure 29 we show a correlation analysis of the algorithm instances' performance in terms of the expected run time measured in evaluated solutions equivalents. There is some notable correlation among the solvers though and easier instances require less effort overall. This is a nice property, nevertheless, there is no clear indication of what is a simple instance. One would assume that this observed correlation is due to problem dimension influencing the algorithm performance of all solvers. However, dimension only explains some variation as the first column in Figure 29 shows and there are instances as small as size 19 which are very hard for some instances as well as those of sizes

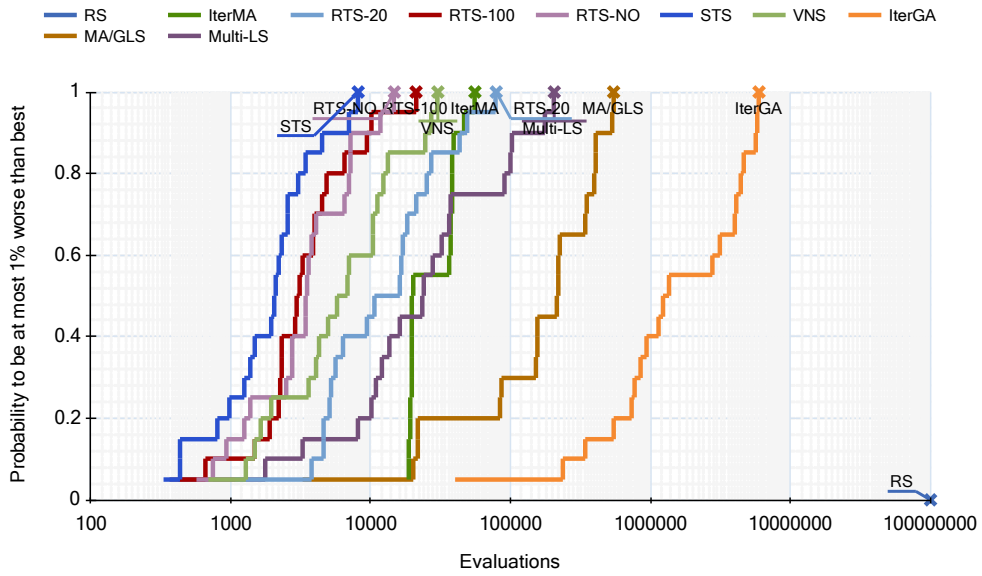



Figure 28: Runtime analysis of different algorithm instances for the 1% target on the nug30 problem instance.

close to 100 that are still rather easy to solve for some. The figure shows on the lower left diagonal a scatter plot of the $\log_{10}(\text{ERT})$ performance values where ERT stands for *expected run time*. On the upper right diagonal the respective Pearson's correlation coefficient $R^2$ is shown. In the case of an ERT value of $\infty$, $\log_{10}(\text{ERT}) = 10$ was chosen.

### 4.1.5 Recommendation Algorithms

As a recommender we chose a $k$-nearest neighbor-based algorithm (k-NN) which calculates distances among the features using the *L2-norm*. The features are normalized to 0 mean and unit variance in order not to bias the distance towards features with greater range. Normalization parameters are estimated from training instances only and applied to the new problem instances.

K-nearest neighbor algorithms are instance-based algorithms that do not require training. Its performance is influenced by the hyperparameter $k \in \mathbb{N}$ that can be said to control the generalization. The higher $k$ the more instances will be considered to determine the regression value or class label. But also the set of features that comprise distance has a strong influence on the performance of the recommender.

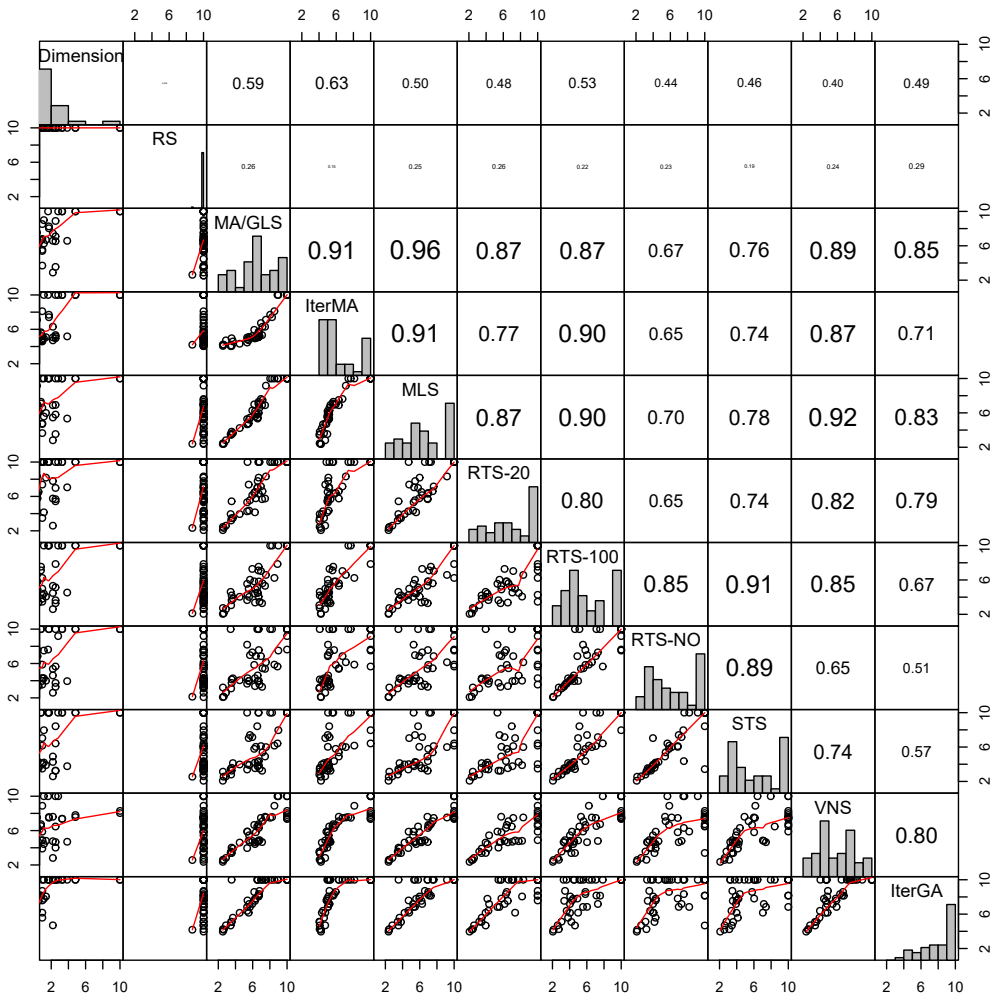



Figure 29: Correlation analysis of the algorithm instances' expected runtime measured in number of evaluations and given as $\log_{10}(\text{ERT})$ when applied to the benchmark data set. The problem dimension has been linearly rescaled to fit into the range from 1 to 10.

The output of the recommender should be a ranking of algorithm instances and potentially an estimation on their runtime to achieve a certain target or an estimation of the target given a certain budget. If neither are given it is possible to average between a set of targets and budgets. For this study we will assume that a fixed target is given by the user and algorithms should be recommended that achieve that target in shortest time.

The ranking of each algorithm instance is determined by the observed rankings of the $k$ closest problem instances. In the described experiment, a new ranking will be created by averaging the observed ERT values in each of the problem instances and sorting the algorithm instances by that average ERT. A small expected runtime will lead to a better rank than a higher runtime.

### 4.1.6 Evaluation

We will perform two tests in this study. In the first test, we will evaluate the quality of the exploratory landscape approach to identify the correct problem instance similar to the tests performed in Section 3.3.2 on page 80. In this test we aim to identify how much effort has to be spent such that the features uniquely identify a certain problem instance *within the chosen benchmark set* (similar to the analysis performed in Section 3). Of course it is assumed that this becomes better the more samples we obtain. In the second test, the performance of the recommender that is applied to solve the algorithm selection problem is evaluated on the set of benchmark instances using leave-one-out (LOO) crossvalidation. Thereby, each of the instances is excluded from the training set and treated as if it was previously unknown. A ranking and recommendation is then obtained given the other instances and the observed performance. This ranking is then compared to the actual ranking that was observed in the test instance. Repeating this for all instances then gives us an estimator on the quality of the recommender.

As has been mentioned before we observe a clustering of algorithms' performances. To a user, it is important to obtain one algorithm out of that cluster, but not exactly which. Thus, the algorithm instances will be clustered into six classes for each problem instance. Given the assumption of a pre-defined target quality by the user, the clustering is concerned with the runtime performance dimension. There exist efficient clustering algorithms such as Ckmeans.1d.dp for 1-dimensional k-Means that we employ in this study [WS11a]. Again, we use the $\log_{10}(\texttt{ERT})$ performance values [HS98, AH05]. The log-transformation

is more akin to the users' needs, because the difference in log values translates to a quotient of the actual values. Thus, the clustering considers distances which can be interpreted as "n times faster" (cf. Section 2.5 on page 47). If however the user fixed the budget instead of the target, then the clustering would be performed on the performance measure in the solution quality domain for which an appropriate transformation is a priori more difficult to justify. A log-transformation might not be desired in such a case and may not even be mathematically possible, e.g. in case of non-positive quality values. In this study it is assumed that only the target quality value is fixed.

After Ckmeans.1d.dp is finished the clusters are sorted by their centroids and the class values 1 to 5 are assigned in this order. In the obtained ranking class 1 holds the best algorithm instances, class 2 the $2^{\text{nd}}$ best instances and so on. Class 6 is special and contains algorithm instances that failed to achieve the x% target quality in any run. Their ERT would be $\infty$ and they are thus put in their own class. While algorithm instances from class 1 to 5 are generally usable, but with decreasing efficiency, algorithm instances of class 6 are unsuitable and should not be recommended.

We attempt to evaluate the recommender algorithm in this described set-up. The $n$ best-ranked algorithm instances for each problem instance are determined and compared with the actual rankings. It has to be mentioned however, that for some problem instances only a single algorithm instance achieved the target quality. Thus for a setting of $n > 1$ the recommender includes unsuitable instances in its suggestion.

### 4.1.7 Recommendation Performance Measures

We use the normalized discounted cumulative gain (NDCG) and Kendall's $\tau$ to evaluate the performance of the proposed recommender. NDCG [JK02] is a measure from the field of information retrieval. In our case, it describes the reward or gain of having a certain algorithm instance in a certain rank. Lower ranked algorithm instances are discounted and do not provide the same gain as a higher ranked document. The normalization is performed by dividing the observed discounted cumulative gain (DCG) with the optimal or ideal discounted cumulative gain (iDCG). The latter would rank each document exactly as it should be ranked according to maximize gain. This results in a number between 0 (bad ranking performance) and 1 (good performance). The NDCG is often limited to the highest ranked $n$ documents and then denoted

as $\text{NDCG}_n$. Translated to the recommendation of algorithm instances, the documents correspond to the instances and the observed performance is the basis for the desired ranking.

Kendall's $\tau$ [Ken38] is a rank correlation value. It considers a combination of all unique pairs $(x_i, y_i)$ and $(x_j, y_j)$ in two rankings $X$ and $Y$ and is described as the ratio between the difference of "agreeing" and "disagreeing" pairs and the total number of pairs. In an agreeing pair, both $x_i$ and $y_i$ are ranked higher or lower than $x_j$ and $y_j$, otherwise it is considered a disagreeing pair. Kendall's $\tau$ falls within the range $[-1; 1]$ where -1 indicates two rankings are exactly opposite and 1 indicates they are exactly equal. A value of $\tau = 0$ indicates no correlation.

### 4.1.8 Analysis of the Accuracy of Exploratory Landscape Features

In the first part of the results we show the ability to correctly re-identify the problem instances in a similar way as was shown in Section 3.3.2 on page 80. This analysis should answer the question of how many effort needs to be devoted to (offline) landscape analysis in order to achieve a robust estimate of its features. The robustness in this case is analyzed in terms of the average rank that a certain landscape is observed in when comparing the Euclidean distance of the feature vector obtained during training compared to that obtained during test. Due to the stochastic method of exploratory landscape anaylsis each time a slightly different feature vector is obtained.

In Table 12 we evaluate the ranking based on a baseline of 200 paths. This means that we obtain the features sharpness, bumpiness, and flatness (introduced in Section 3.2 on page 71) of all involved problem instances in the study using 200 paths (training set). Then we take each problem instance and compute the features again using different seeds and rank the training set according to the Euclidean distance to the new vector. The obtained rank is recorded. Ideally, this rank has to be 1, but when instances are more difficult to identify another instance might be closer and thus a higher rank may be observed. We produce an average over all problem instances and thus obtain the strength of the number of paths involved. This process is then repeated 5 times and again an average is obtained. The results are shown in Table 12.

The results in Table 12 show that the given problem instances are discernible using the features and given about 100 paths which is still computationally challenging.

Table 12: The average rank over 5 repetitions is shown using the three features sharpness, bumpiness, and flatness with a training effort of 200 paths and a varying test effort as shown in the columns. [BAW17]

| | **Paths** | | | | | |
|---|---|---|---|---|---|---|
| | **5** | **10** | **20** | **50** | **100** | **200** |
| Average Rank | 8.2 | 7.3 | 4.6 | 2.7 | 1.9 | 1.4 |

### 4.1.9 Analysis of a k-Nearest Neighbor Recommender

The results of the recommendation measures are given in Table 13. It can be seen that good $\texttt{NDCG}_n$ values can be achieved. The table shows the recommender performance for different relative target values. The values vary somewhat between different targets and a simple progress may not be observed. Rankings need however not be stable across the target values. An algorithm that is first to reach target X may reach target Y rather late. Additionally, a closer target such as 1% or even the optimum at 0% deviation can lead to situations where most algorithm instances fail and the performance classes become densely populated. Target values closer to the optimum lead to harder recommendation problems as the number of algorithm instances that may achieve those targets becomes much smaller. The results in Table 13 suggest that for recommending three algorithm instances ($\texttt{NDCG}_3$) good recommendations can be achieved for a 5% target. For this target we observe that there were 18 (37.5%) cases where only a single algorithm instance was in rank 1.

Figure 30 shows a comparison of different algorithm instances as they performed on the benchmark instances, while Figure 31 shows the performance classes of RTS without aspiration with varying target levels from 0% (optimum) to 10% deviation. The results show that there is one range of instances (the taiXXe instances) that are not suitable for RTS without aspiration. As Figure 30 shows these may be solved quite well with VNS. The combination of RTS and VNS seems to form a successful team where one solves instances that the other is not so good at and vice versa.

A detailed and closer look at the exact recommendation when three algorithm instances are to be recommended for the just mentioned setting is given in Table 14. The results indicate that the recommender can achieve the best possible recommendation for 26 out of 47 problem instances (55%). In the remaining 21 problem instances, algorithm instances of class 6 were part of the

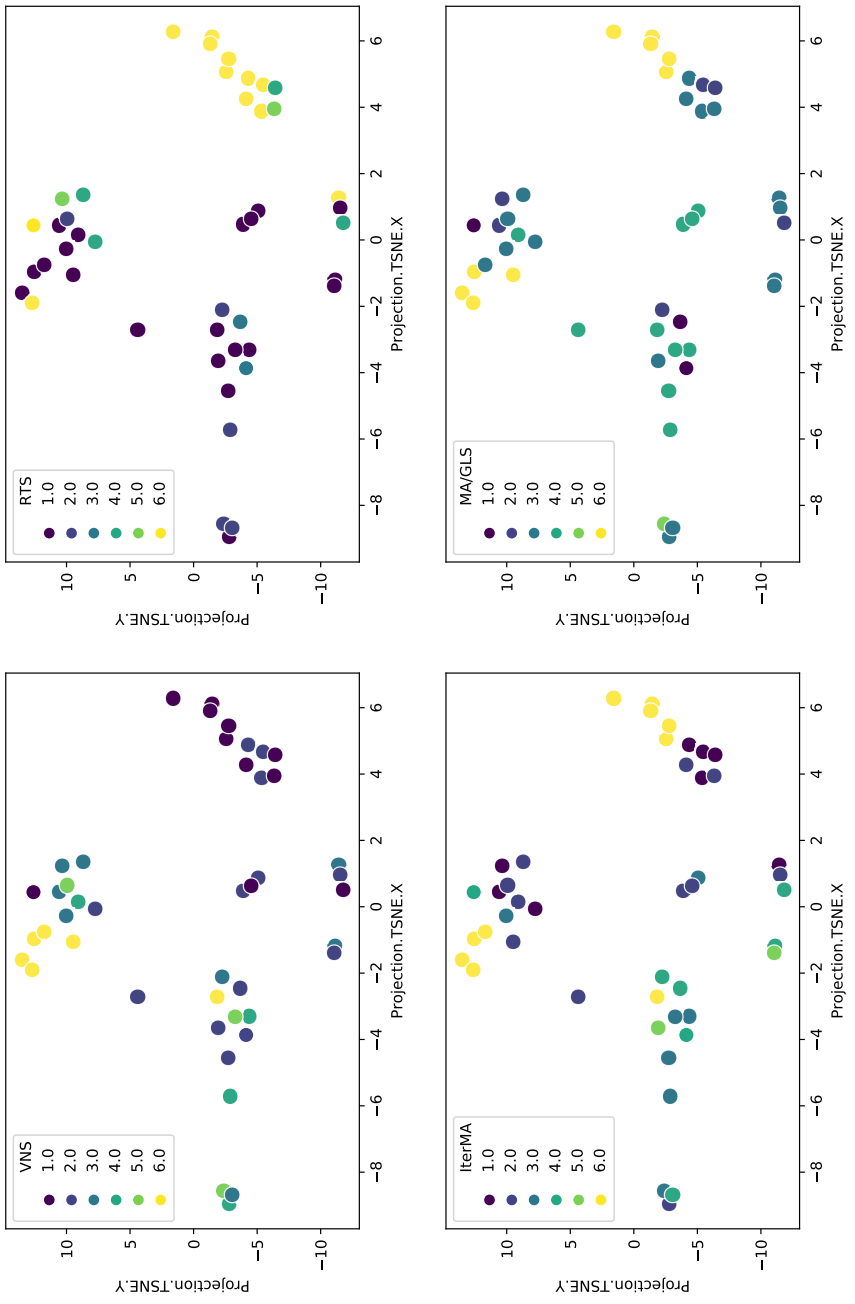



Figure 30: Performance classes for the 1% target for VNS, RTS-noasp, IterMA, and MA/GLS (left to right, top to bottom). Each dot represents a problem instance projected from problem specific feature space with t-sne.

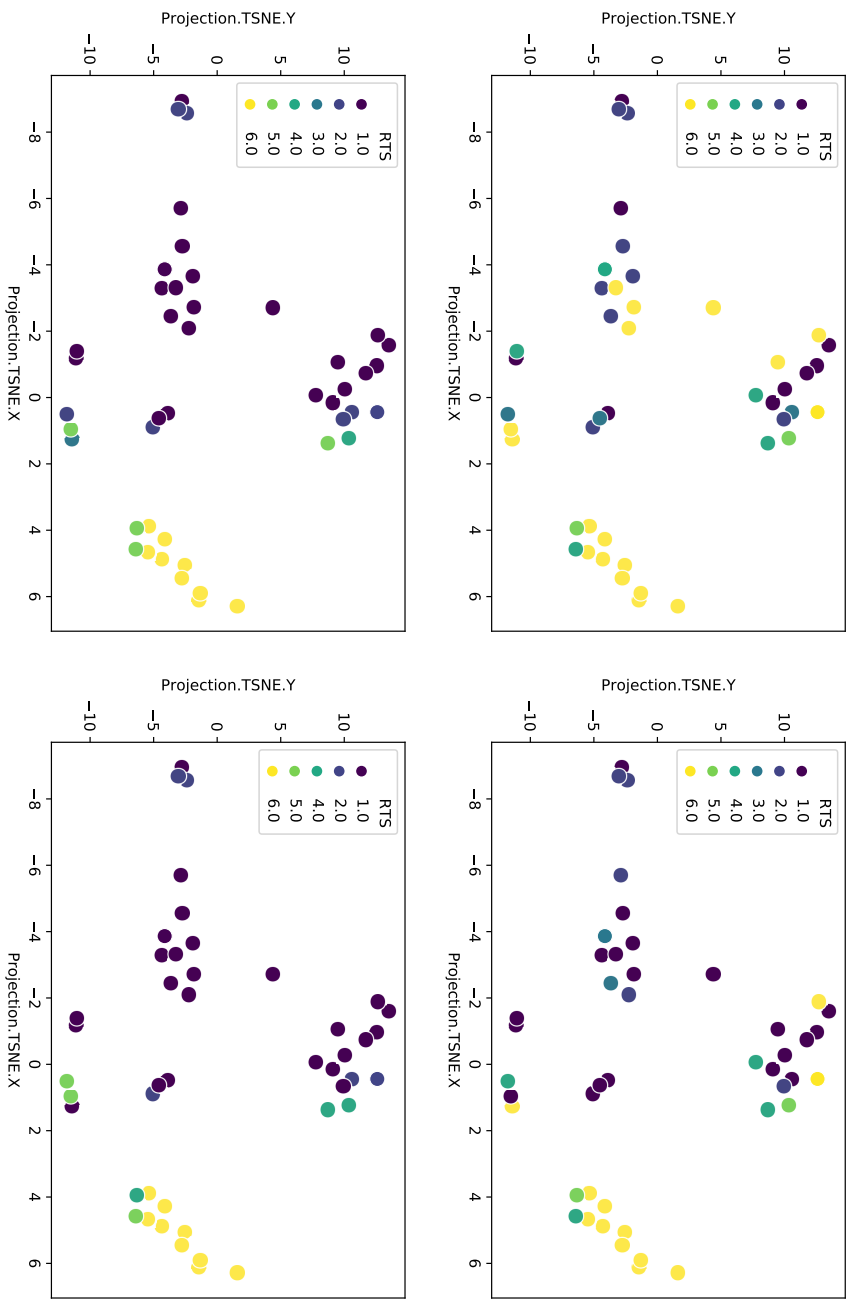



Figure 31: Performance classes for the 0%, 1%, 5%, and 10% target for RTS-noasp (left to right, top to bottom). Each dot represents a problem instance projected from problem specific feature space with t-sne.

recommendation only for three instances (*dre56*, *lipa50b*, *tai125e10*). These algorithm instances are unsuitable because they were not able to achieve the desired target level within 100,000,000 evaluations in any of the observed runs. In other cases unsuitable instances had to be recommended because there were less than three suitable algorithm instances. Also, in the 21 problem instances that are not recommended perfectly, no algorithm instance of class 1 was recommended only three times (*had20*, *tai20b*, *tai100b*). For 18 problem instances, and thus in total for 44 out of 47 (93.6%) problem instances, algorithm instances of class 1 were part of the recommendation.

Analyzing the correlation of all recommended algorithm instances we have to turn to Kendall's $\tau$ in Table 13. The problem specific characteristics achieved the best possible rankings, with second being the features introduced in Section 3 and third being that of random walks with 10,000 iterations.

In the best possible scenario as shown in Table 13 we take a closer look at the performances of each individual solver in comparison to the combined solver using a one nearest neighbor recommender. Table 15 shows the relative number of problem instances for which each algorithm instance achieved which rank. As can be seen the recommender for this target performs very well and better than any of the individual solvers both in the number of instances where it achieves rank 1 and the number of times that it failed. For the harder 1% target we can observe in Table 16 that the combined, portfolio-based approach still outperforms any individual solver. Again the problem specific features have been used.

In Figure 32 we observe the best performances of algorithm instances in the QAP benchmark set. Each dot represents a problem instance and its associated best performing algorithm instances - in case multiple dots are stacked inside each other.

Table 13: The resulting performance measures evaluated for a varying number of ranked algorithm instances. The number $n$ denotes that performance measure if only the top $n$ ranked algorithm instances are taken into account. [BAW17]

| Target | $\texttt{NDCG}_n$ | | | | | | Kendall's $\tau$ |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | |
| Sharpness, Bumpiness, Flatness (200 paths) | | | | | | | |
| 0% | 0.78 | 0.84 | 0.87 | 0.89 | 0.90 | 0.90 | 0.65 |
| 1% | 0.80 | 0.80 | 0.85 | 0.85 | 0.86 | 0.87 | 0.61 |
| 5% | 0.86 | 0.85 | 0.88 | 0.90 | 0.90 | 0.90 | 0.65 |
| 10% | 0.83 | 0.85 | 0.89 | 0.91 | 0.91 | 0.91 | 0.61 |
| 20% | 0.80 | 0.81 | 0.86 | 0.87 | 0.88 | 0.89 | 0.58 |
| Dimension, Sharpness, Bumpiness, Flatness (200 paths) | | | | | | | |
| 0% | 0.80 | 0.86 | 0.88 | 0.90 | 0.90 | 0.90 | 0.62 |
| 1% | 0.82 | 0.83 | 0.86 | 0.86 | 0.87 | 0.88 | 0.62 |
| 5% | 0.88 | 0.87 | 0.89 | 0.90 | 0.90 | 0.91 | 0.65 |
| 10% | 0.86 | 0.87 | 0.91 | 0.92 | 0.92 | 0.91 | 0.61 |
| 20% | 0.80 | 0.83 | 0.88 | 0.89 | 0.89 | 0.89 | 0.57 |
| QAP specific characteristics | | | | | | | |
| 0% | 0.77 | 0.79 | 0.83 | 0.86 | 0.86 | 0.87 | 0.67 |
| 1% | 0.77 | 0.80 | 0.83 | 0.85 | 0.87 | 0.88 | 0.63 |
| 5% | 0.95 | 0.94 | 0.93 | 0.94 | 0.94 | 0.94 | 0.72 |
| 10% | 0.93 | 0.92 | 0.93 | 0.93 | 0.93 | 0.94 | 0.70 |
| 20% | 0.89 | 0.90 | 0.92 | 0.92 | 0.92 | 0.92 | 0.68 |
| Random walk characteristics (10,000 iterations) | | | | | | | |
| 0% | 0.72 | 0.77 | 0.79 | 0.81 | 0.82 | 0.83 | 0.53 |
| 1% | 0.69 | 0.72 | 0.74 | 0.76 | 0.78 | 0.79 | 0.52 |
| 5% | 0.86 | 0.84 | 0.84 | 0.86 | 0.86 | 0.87 | 0.57 |
| 10% | 0.86 | 0.85 | 0.86 | 0.87 | 0.89 | 0.89 | 0.59 |
| 20% | 0.83 | 0.82 | 0.84 | 0.85 | 0.87 | 0.87 | 0.56 |

Table 14: Result of the recommendation. In each cell "X/Y" indicates that X out of Y algorithms in that class have been recommended. Bold problem instances indicate the best recommendation has been achieved. [BAW17]

| **Problem Instance** | **Performance Classes 5% target** | | | | | |
|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** |
| **bur26a** | **3/4** | -/2 | -/2 | -/1 | -/1 | |
| **bur26d** | **3/4** | -/3 | -/1 | -/1 | -/1 | |
| chr20a | 2/3 | 1/2 | -/2 | -/1 | -/1 | -/1 |
| chr20b | 1/1 | 1/2 | 1/2 | -/3 | -/1 | -/1 |
| chr20c | 1/3 | 1/2 | -/2 | -/1 | 1/1 | -/1 |
| **dre24** | **3/3** | -/1 | -/1 | -/2 | -/1 | -/2 |
| **dre30** | **2/2** | **1/1** | -/3 | -/1 | -/1 | -/2 |
| dre56 | 1/1 | -/1 | -/1 | | | 2/7 |
| **dre72** | 1/1 | | | | | 2/9 |
| **dre110** | 1/1 | | | | | 2/9 |
| els19 | 1/3 | 1/2 | -/1 | -/2 | 1/1 | -/1 |
| esc32a | 1/1 | 1/4 | -/1 | -/1 | 1/2 | -/1 |
| had20 | -/4 | 2/3 | -/1 | 1/1 | -/1 | |
| kra32 | 2/2 | -/3 | 1/2 | -/1 | -/1 | -/1 |
| **lipa20a** | **3/4** | -/2 | -/1 | -/2 | -/1 | |
| **lipa20b** | **1/1** | **2/3** | -/3 | -/1 | -/1 | -/1 |
| lipa50a | 2/6 | -/1 | 1/2 | -/1 | | |
| lipa50b | 2/3 | -/2 | -/1 | -/1 | -/1 | 1/2 |
| **lipa90a** | **3/6** | -/1 | -/2 | -/1 | | |
| **lipa90b** | **1/1** | **2/3** | -/1 | -/1 | -/2 | -/2 |
| nug30 | 1/2 | 1/2 | 1/3 | -/1 | -/1 | -/1 |
| RAND-S-6x6-...bl | 2/3 | -/2 | 1/2 | -/1 | -/1 | -/1 |
| **RAND-S-8x8-...ci** | **3/4** | -/2 | -/1 | -/1 | -/1 | -/1 |
| RAND-S-10x10-...bl | 1/4 | -/1 | 1/1 | -/1 | 1/1 | -/2 |
| **RAND-S-12x12-...ci** | **3/4** | -/2 | -/1 | -/1 | -/1 | -/1 |
| **sko56** | **3/4** | -/2 | -/1 | -/1 | -/1 | -/1 |
| **sko90** | **3/4** | -/2 | -/1 | -/1 | -/1 | -/1 |
| **tai20a** | **3/4** | -/2 | -/1 | -/1 | -/1 | -/1 |
| tai20b | -/1 | 1/2 | 1/3 | 1/3 | -/1 | |
| **tai50a** | **3/4** | -/1 | -/2 | -/1 | -/1 | -/1 |
| tai50b | 1/3 | 1/2 | 1/2 | -/1 | -/1 | -/1 |
| **tai100a** | **3/4** | -/2 | -/1 | -/1 | -/1 | -/1 |
| tai100b | -/2 | 1/3 | 1/1 | -/1 | 1/2 | -/1 |
| **tai27e01** | **2/2** | **1/2** | -/2 | -/1 | -/2 | -/1 |
| **tai27e10** | **2/2** | **1/1** | -/2 | -/1 | -/3 | -/1 |
| tai45e01 | 1/1 | 1/2 | 1/1 | -/1 | -/1 | -/4 |
| tai45e10 | 1/1 | 1/2 | 1/1 | -/1 | -/1 | -/4 |
| tai75e01 | 1/1 | 1/1 | -/1 | 1/1 | | -/6 |
| tai75e10 | 1/1 | 1/1 | -/1 | 1/1 | | -/6 |
| **tai125e01** | **1/1** | **1/1** | | | | **1/8** |
| tai125e10 | 1/1 | -/1 | | | | 2/8 |
| **tai175e01** | **1/1** | **1/1** | | | | **1/8** |
| **tai175e10** | **1/1** | | | | | **2/9** |
| **tai343e01** | **1/1** | | | | | **2/9** |
| **tai343e10** | **1/1** | | | | | **2/9** |
| **wil50** | **3/4** | -/2 | -/1 | -/1 | -/1 | -/1 |
| **wil100** | **3/4** | -/2 | -/1 | -/1 | -/1 | -/1 |
| Recommended | 80 | 24 | 11 | 4 | 5 | 17 |
| Total | 119 | 79 | 59 | 45 | 40 | 128 |

Table 15: The combined solver given the one nearest neighbor achieved better performance than the individual solvers for the 5% target.

| Alg.Inst | $1^{st}$ | $2^{nd}$ | $3^{rd}$ | $4^{th}$ | $5^{th}$ | $6^{th}$ |
|---|---|---|---|---|---|---|
| **Combined** | 92% | 2% | 2% | 0% | 2% | 2% |
| STS | 58% | 4% | 0% | 0% | 10% | 25% |
| RTS-NOASP | 50% | 15% | 2% | 2% | 8% | 21% |
| RTS-100 | 44% | 17% | 6% | 10% | 2% | 19% |
| RTS-20 | 35% | 15% | 15% | 4% | 0% | 29% |
| VNS | 25% | 38% | 21% | 6% | 2% | 6% |
| IterMA | 17% | 17% | 10% | 25% | 17% | 12% |
| MLS | 12% | 44% | 6% | 8% | 4% | 23% |
| MA/GLS | 6% | 15% | 50% | 8% | 2% | 17% |
| IterGA | 0% | 0% | 6% | 27% | 33% | 31% |

Table 16: The combined solver given the one nearest neighbor achieved better performance than the individual solvers for the 1% target.

| Alg.Inst | $1^{st}$ | $2^{nd}$ | $3^{rd}$ | $4^{th}$ | $5^{th}$ | $6^{th}$ |
|---|---|---|---|---|---|---|
| **Combined** | 53% | 23% | 4% | 2% | 2% | 15% |
| RTS-NOASP | 42% | 10% | 4% | 8% | 4% | 29% |
| STS | 40% | 10% | 6% | 0% | 8% | 33% |
| RTS-100 | 29% | 23% | 6% | 6% | 4% | 29% |
| VNS | 25% | 27% | 17% | 8% | 6% | 15% |
| IterMA | 19% | 23% | 15% | 15% | 4% | 23% |
| RTS-20 | 15% | 19% | 6% | 8% | 8% | 42% |
| MA/GLS | 6% | 12% | 33% | 21% | 2% | 23% |
| MLS | 4% | 23% | 21% | 10% | 6% | 33% |
| IterGA | 0% | 0% | 0% | 12% | 38% | 48% |

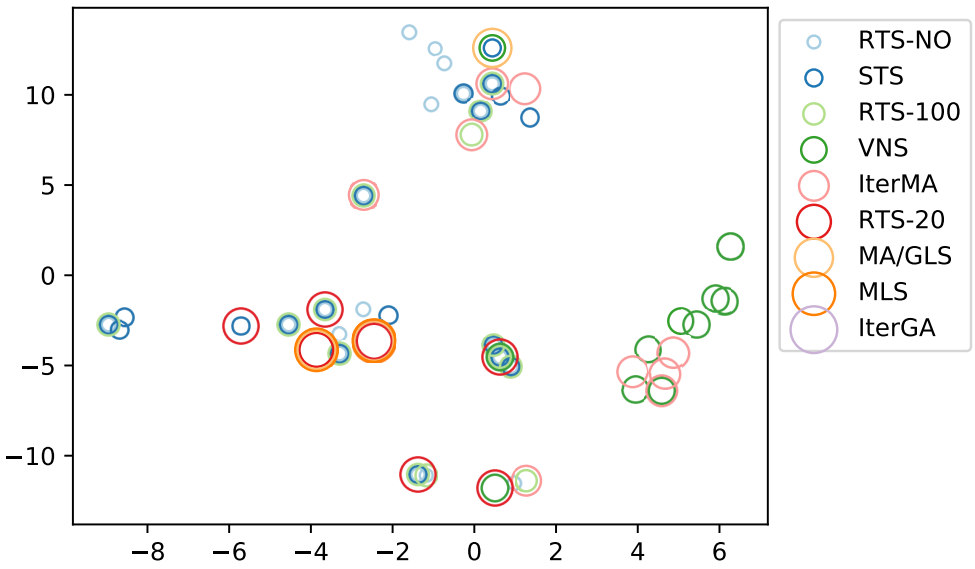



Figure 32: Problem instance map showing the best performing algorithm instance(s) for the 1% target. Each dot represents a problem instance projected onto a plane with t-sne.

### 4.1.10 Conclusions

We have performed an algorithm selection case study on a range of diverse instances of quadratic assignment problems. We have chosen a set of metaheuristic algorithms for solving the problem and analyzed their performance. In particular, we looked at correlations in the performance among solvers. We looked at the results in more detail and identified a good combination in a portfolio that includes RTS and VNS. We have also evaluated a nearest neighbor recommender among all algorithm instances and evaluated its performance in a leave-one-out crossvalidation setting. The results indicate that a portfolio-based approach can be more successful than a single solver.

As we have outlined at the very beginning, we consider each algorithm instance as a unique entity. However, for future work it might be interesting to also consider an embedding of algorithm instances into a vector space of its own and identify whether there are relations between those spaces, which could lead to new insights and discoveries.

## 4.2 Algorithm Selection for Solving Generalized Quadratic Assignment Problems

In this section I present an extension of the work that has been presented at GECCO 2018 [BWA18] and which has been done by myself, Stefan Wagner, and Michael Affenzeller. The main contributions of this part to the state of the art are:

1. Performing landscape analysis on the generalized quadratic assignment problem (GQAP)
2. Benchmarking of algorithms and solvers, including open source implementations of metaheuristic algorithms as well as commercial-of-the-shelf solvers
3. Evaluation of algorithm instance recommendation based on k-nearest neighbor classifiers

The work described by Beham et al. [BWA18] was distributed such that Beham devised the methodological approach and scope, devised and implemented the algorithm and problem instances used in the study (if not otherwise stated), and carried out the experiments. Beham implemented and carried out the landscape analysis and generated, and analyzed the algorithm selection results. Affenzeller and Wagner both contributed with comments, engaged in discussions, and supported the editing process of the final text.

The generalized QAP is a related, but also notably different problem from the QAP. I have introduced the GQAP more formally already in Section 2.1.4. In a brief summary of the differences, the size of the solution space grows distinctly and amounts to $M^N$ in contrast to the solution space of the QAP ($N!$). Thus, there can be both larger and smaller instances of the comparable QAP (same $N$) depending on the value of $M$. The assignment encoded in a solution to the GQAP is typically represented as a vector of integers where the binary matrix $x_{ik}$ is replaced by the integer vector $y[i] \leftarrow k$ where $x_{ik} = 1$. This vector may also never violate the constraint given in Equation (2.13). But, unlike the QAP that encodes solutions as permutations, the general discrete solution space requires different heuristic operators. Crossovers and manipulation or perturbation operators defined on permutations are not feasible to be employed in the context of the integer vector. A further difference between the QAP and the GQAP is the presence of constraints such as the capacity restriction which turns some parts of the solution space infeasible. To handle

such a constrained optimization problem several techniques are described in the literature [MMC11] for which we chose to go with a lexicographic approach. First the constraint violation is minimized and then the objective. Thus, we generally favor feasible solutions over infeasible ones. In the category of infeasible solutions we prefer solutions that violate the constraints only slightly to those that violate them to a larger degree, while among the feasible solutions the better ones according to the objective are preferred of course.

We achieved the lexicographic approach using a single fitness value that is split into two separated domains. The infeasible solutions were offset by a penalty as given in Equation (4.4). This penalty is higher than the quality of any feasible solution. In addition we add the amount of capacity that a solution overuses to that penalty in order to comprise the fitness of an infeasible solution. Feasible solutions are valued the same as the objective given in Equation (2.11) on page 13. The effect is similar to other proposed constraint handling techniques [Kal00].

$$P = \max(C) \cdot N + e \cdot \max(W) \cdot \max(D) \cdot N^2 \tag{4.4}$$

### 4.2.1 Benchmark Instances

There are benchmark instances already available which comprise the first 21 benchmark instances [CGLM06] of this case study ranging from 20 to 50 facilities and 6 to 20 locations. The utilization, as computed by Equation (4.5) ranges from 35% to 95% and denotes the ratio between total demand and total capacity as shown. Utilization determines the size of the feasible search space and a high utilization leads to many more solution configurations that are infeasible.

$$\texttt{util.:} \quad \frac{\sum_{t \in 1..N} Q_t}{\sum_{r \in 1..M} C_r} \tag{4.5}$$

In addition to those instances we generated new instances based on the well-known QAPLIB [BKR97]. For each instance in this library where the dimension $N \geq 20$ we produce three groups of instances with different number of locations. These have been chosen as the rounded value of $M = \{N/6, N/3, 2 \cdot N/3\}$. The matrix $D$ that describes the "distance" between locations is generated by running a hierarchical clustering on the QAPLIBs'

instance's original distances and averaging the distances between the clusters. Thus, it is ensured that structures present in the distances of locations of the QAP instance are translated to the GQAP instance. Because instances in the QAPLIB do not define a linear cost matrix, we generated the installation costs $C$ randomly by sampling from a uniform distribution in the interval shown in Equation (4.6).

$$C_{ik} \in [1, N \cdot \overline{W} \cdot \overline{D}) \quad \forall i \in \{1, ..., N\} \quad \forall k \in \{1, ..., M\} \tag{4.6}$$

Here, $\overline{W}$ and $\overline{D}$ describe the mean of the weights and distance matrix respectively. The demands $R_t$ were sampled independently from a uniform distribution in the half-open interval $[1, 100)$. Each of the three groups with the number of locations fixed is then split into four additional groups with utilizations $U = \{35\%, 50\%, 75\%, 95\%\}$. Because the demands were already sampled, we achieved this by determining the capacity of each location. First the total capacity $\hat{B}_u$ was computed for utilization group $u$ and was then randomly split among the locations $g$ ($B_{gu}$) adhering to the constraints given in Equations (4.7) to (4.11). These constraints ensure, that a large number of the generated instances have at least one feasible solution on the one hand. On the other hand, the constraint given in Equation (4.11) ensures that not all of the facilities can be assigned to a single location. Whether such instances are more or less difficult has however not been analyzed in more detail, but it is likely that a more carefully constructed installation costs matrix $C$ would be required. Otherwise if the optimal solution would be to put all facilities in that location it may be rather easy to find. A rejection sampling technique was used to satisfy the constraints.

The way we generate $C$, $R$, and $B$ is similar to existing generators [CGLM06], however the new instances provide a much more diverse set with respect to the different matrices for $W$ and $D$. From the generated instances we took 168 for the presented study. The generated instances have been made available online[7].

[7] dev.heuristiclab.com/AdditionalMaterial#GECCO2018

$$\hat{B}_u = \frac{1}{u} \cdot \sum_{t \in 1..N} R_t \qquad \forall u \in U \tag{4.7}$$

$$\hat{B}_u = \sum_{g \in 1..M} B_{gu} \qquad \forall u \in U \tag{4.8}$$

$$B_{gu} \geq \min_{t \in 1..N} (R_t) \qquad \forall g \in 1..M, u \in U \tag{4.9}$$

$$\max_{g \in 1..M} (B_{gu}) \geq R_t \qquad \forall t \in 1..N, u \in U \tag{4.10}$$

$$B_{gu} < \sum_{t \in 1..N} R_t \qquad \forall g \in 1..M, u \in U \tag{4.11}$$

### 4.2.2 Feature Extraction

The first step in the algorithm selection methodology is to define and compute features of the problem instances. This process involves still manual work, though the methods described in Section 3 are somewhat more general. For instance, to perform random, adaptive, or directed walks the neighborhood function needs to be defined. In the case of directed walks this neighborhood function requires that any solution can be transformed into any other through a finite sequence of steps. However, as we have already observed in the last section problem specific features may work quite well and these require to be redefined.

In the case of the GQAP we use the features that have already been defined in the respective publication [BWA18] and which are repeated here for the sake of completeness:

- Problem specific: dimension ($|N|$), MN-ratio ($|M|/|N|$), the coefficient of variation for $W$, $D$, $B$, $R$, the sparsity for $W$, $D$ ($W_0, D_0$), utilization, and basic feasibility
- Random walk: autocorrelation (ac1), correlation length (c.len), information content (ic), partial information content (pic), density basin information (dbi), information stability (is), diversity, regularity, total entropy ($H(X)$), peak information content (ic$^*$), and peak density basin information (dbi$^*$).
- Directed walk: sharpness, bumpiness, flatness

We use sparsity in that we calculate the ratio between the amount of entries that are zero and the size of the matrix. Basic feasibility is specific to GQAP

being a constrained optimization problem and describes the percentage of valid isolated assignment pairs and is given in Equation (4.12).

$$\texttt{feas.:} \quad \frac{\sum_{t \in N} |\{g \in M | R_t \leq B_g\}|}{N \cdot M} \tag{4.12}$$

Random walk-based landscape measures that have been described in the literature are also applied [AZ98, VFM00, PA12]. In this study a walk length of 5,000 is used and the results of 30 independently performed walks is averaged. For the directed walks again we perform a total of 200 paths.

### 4.2.3 Fitness Landscape Analysis

After performing the landscape analysis by applying the walks and calculating the problem specific calculations we obtain a feature vector that describes each problem instance. In Table 17 we study those features in terms of their correlations. This presents a picture in which we can observe to which extent two features respond similarly to the properties of the landscape. Naturally, this is biased by the selection of the benchmark instances and does not represent a view of the GQAP overall, but of the chosen set of instances.

By looking at the problem specific features such biases can be highlighted. For instance, we observe a highly significant negative correlation ($\rho \approx (-0.57, -0.52), p < 1e^{-6}$) between the sparsity of the weights and distance matrices ($W_0$, $D_0$) respectively and the dimension ($|N|$). As the dimension grows, the sparsity reduces which shows that there is a certain bias in that larger instances are becoming less sparse. Another significant correlation is observed between $M/N$ and *feas.* ($\rho \approx -0.57, p < 1e^{-6}$). It is clear that by increasing the number of locations $M$ the capacity per location becomes lower and thus the chance is higher that there are facilities with a higher demand than some of the locations. Still, given the moderate strength of the correlation this is not fully determined and variation does exist. But we also see correlation between $M/N$ and $D_0$ ($\rho \approx -0.64, p < 1e^{-6}$). Thus, a higher number of locations relative to the number of facilities is observed together with distance matrices that are less sparse. The same is not true for $W_0$ however. This may be explained with respect to the zero entries in the distance matrices' diagonal.

By examining the correlation among the three features CV($B$), *flat.*, and *feas.* it can be seen that both *flat.* and CV($B$) are negatively correlated to *feas.* ($\rho < -0.5, p < 1e^{-6}$) while the correlation between *flat.* and CV($B$) are

Table 17: Correlations among FLA characteristics using Spearman's $\rho$. Significant correlations on the upper diagonal: *** indicate a p-value $< 1e^{-6}$, ** for $p < 1e^{-3}$, * for $p < 0.05$. Bonferroni adjustment is used. [BWA18]

| | feas. | $CV(B)$ | $CV(R)$ | $\lvert N\rvert$ | $CV(D)$ | $D_0$ | $CV(W)$ | $W_0$ | $M/N$ | util. | bump. | flat. | sharp. | ac1 | c.len | dbi | div. | ic | is | pic | dbi* | ic* | reg. | $H(X)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| feas. | 1 | *** | | | | *** | | | *** | * | *** | *** | *** | *** | ** | | * | * | *** | | | | *** | * |
| $CV(B)$ | -0.54 | 1 | | | | | | | | | *** | *** | | | | | | | | | | | | |
| $CV(R)$ | -0.05 | 0.07 | 1 | | | | ** | | | | | | | | | | | | | * | | | | |
| $\lvert N\rvert$ | -0.15 | 0.18 | 0.11 | 1 | | *** | | *** | | | | *** | *** | | | | *** | *** | | | | | ** | *** |
| $CV(D)$ | -0.17 | -0.04 | -0.19 | 0.25 | 1 | ** | | | ** | | | | | | | | | | | | | | | |
| $D_0$ | 0.44 | -0.01 | 0.1 | -0.52 | -0.36 | 1 | | *** | *** | | ** | *** | ** | | | | | * | | | | | | * |
| $CV(W)$ | 0 | 0.1 | -0.36 | -0.01 | 0.03 | -0.21 | 1 | | | | | | | | | | | | | | | | | |
| $W_0$ | -0.08 | 0.19 | 0 | -0.57 | 0.07 | 0.49 | -0.04 | 1 | | | | | | * | | | | | | | | | | |
| $M/N$ | -0.57 | -0.03 | -0.08 | -0.01 | 0.41 | -0.64 | 0 | -0.06 | 1 | | *** | *** | | | | | ** | | ** | | | | *** | |
| util. | -0.31 | -0.06 | -0.08 | 0.03 | 0.1 | -0.02 | -0.03 | -0.05 | 0.01 | 1 | *** | | | | | *** | | *** | *** | *** | *** | *** | * | *** |
| bump. | 0.47 | -0.43 | -0.15 | -0.16 | -0.12 | 0.37 | -0.08 | 0.04 | -0.44 | 0.44 | 1 | *** | *** | * | | ** | | | | ** | ** | ** | | |
| flat. | -0.56 | 0.47 | 0.16 | 0.43 | 0.23 | -0.48 | -0.01 | -0.15 | 0.43 | -0.06 | -0.76 | 1 | *** | * | | | | | | | | | * | |
| sharp. | 0.42 | -0.25 | -0.01 | -0.47 | -0.14 | 0.36 | -0.08 | 0.23 | -0.18 | -0.17 | 0.48 | -0.58 | 1 | *** | *** | | | *** | ** | ** | | | | *** |
| ac1 | -0.44 | 0.19 | 0.15 | 0.23 | -0.24 | -0.24 | -0.2 | -0.3 | 0.2 | 0.19 | -0.3 | 0.3 | -0.49 | 1 | *** | | | ** | *** | | | | | ** |
| c.len | -0.41 | 0.11 | 0.15 | 0.17 | -0.24 | -0.23 | -0.21 | -0.26 | 0.26 | 0.18 | -0.27 | 0.26 | -0.42 | 0.94 | 1 | | | * | *** | | | | | * |
| dbi | 0.13 | 0.26 | 0.12 | 0.11 | -0.1 | 0.01 | 0.04 | 0.03 | -0.06 | -0.73 | -0.4 | 0.07 | 0.11 | -0.07 | -0.07 | 1 | ** | *** | *** | *** | *** | ** | ** | *** |
| div. | 0.29 | 0 | -0.08 | 0.43 | -0.11 | -0.06 | 0.18 | -0.26 | -0.36 | -0.27 | -0.05 | -0.14 | -0.2 | 0.11 | 0.04 | 0.35 | 1 | * | ** | | *** | *** | *** | |
| ic | 0.31 | 0.03 | 0.01 | -0.45 | -0.09 | 0.28 | -0.03 | 0.27 | -0.02 | -0.58 | -0.1 | -0.11 | 0.56 | -0.37 | -0.32 | 0.54 | -0.28 | 1 | *** | *** | *** | *** | | *** |
| is | 0.61 | -0.16 | -0.05 | -0.01 | -0.11 | 0.18 | 0.16 | -0.07 | -0.34 | -0.63 | 0 | -0.21 | 0.38 | -0.52 | -0.48 | 0.53 | 0.37 | 0.47 | 1 | *** | *** | *** | *** | *** |
| pic | -0.15 | -0.28 | -0.06 | 0.2 | 0.07 | -0.14 | 0.01 | -0.19 | -0.02 | 0.68 | 0.35 | -0.12 | -0.35 | 0.23 | 0.21 | -0.75 | 0.1 | -0.91 | -0.47 | 1 | *** | *** | | *** |
| dbi* | 0.16 | 0.25 | 0.06 | 0.13 | -0.13 | 0.01 | 0.09 | 0.01 | -0.12 | -0.68 | -0.37 | 0.06 | 0.13 | -0.02 | -0.03 | 0.84 | 0.5 | 0.47 | 0.52 | -0.65 | 1 | *** | *** | *** |
| ic* | -0.16 | -0.25 | -0.07 | -0.16 | 0.12 | 0.03 | -0.07 | 0.03 | 0.11 | 0.66 | 0.34 | -0.07 | -0.16 | 0.01 | 0.03 | -0.81 | -0.49 | -0.49 | -0.51 | 0.66 | -0.98 | 1 | *** | *** |
| reg. | 0.45 | -0.11 | -0.11 | 0.33 | -0.14 | 0.04 | 0.13 | -0.25 | -0.46 | -0.31 | 0.1 | -0.32 | -0.01 | -0.01 | -0.07 | 0.39 | 0.95 | -0.15 | 0.48 | 0.03 | 0.51 | -0.51 | 1 | |
| $H(X)$ | 0.32 | 0.04 | 0.01 | -0.43 | -0.09 | 0.28 | -0.02 | 0.26 | -0.02 | -0.6 | -0.11 | -0.11 | 0.54 | -0.36 | -0.31 | 0.57 | -0.24 | 1 | 0.48 | -0.92 | 0.51 | -0.52 | -0.12 | 1 |

(necessarily) positively correlated ($\rho \approx 0.47, p < 1e^{-6}$). A stronger variation of the capacities translates into less basic feasibility and this in turn lets the landscape appear more flat. This may be explained by the lexicographic approach to the objective and the high offset for the infeasible solutions. In the infeasible region it is more likely to make neutral choices.

Stronger correlations have also been identified between *util.* and several of the features from exploratory analysis, especially *ic*, *pic*, *dbi*, and $H(X)$) ($\rho \approx (-0.58, 0.68, -0.73, -0.60, p < 1e^{-6})$) and also between *bump.* and *util.* ($\rho \approx 0.44, p < 1e{-}6$), but to a lesser degree. Thus these features are more sensitive to changes in *util.*. The correlation between *feas.* and *util.* was significant, but lower ($\rho \approx -0.31, p < 0.05$). As the utilization increases, the choices for some of the facilities become less as their demand is larger than the capacity of some locations. Still, the effect of *util.* on the landscape is much higher as shown by the correlations with the walk-based landscape features. Other problem specific features were not significantly correlated with *util.*.

### 4.2.4 Algorithm Instances

In this study we applied a wider variety of algorithm instances including open-source implementations as well as commercial solvers, both exact and heuristic and single-solution algorithms as well as population-based methods. Nevertheless, the algorithms and the instances chosen represent only a hand-picked subset of possible algorithms. The selected algorithms are:

1. Iterated Local Search (2 instances)
2. Evolution Strategy (1 instance)
3. Iterated Genetic Algorithm (1 instance)
4. Greedy Randomized Adaptive Search Procedure (1 instance)
5. Late-acceptance Hill Climber (1 instance)
6. Age-Layered Population Structure (1 instance)
7. Linearized Integer Programming (2 instances)
8. Hybrid Mathematical Programming Solver (2 instances)
9. Random Search (1 instance)

These algorithm instances respectively solvers will be described in more detail and with pseudo-code descriptions at least for the open-source implementations. Again we include random search in this study for sake of completeness and as a baseline.

**Iterated Local Search**

As has been described in Section 2.2.4 on page 30 iterated local search is a more general framework that is based on applying an efficient low-level local search to certain points in the landscape. While the general framework as given in Algorithm 2 (page 30) enables the use of history, in the implementation used in the following study this history was not used. A pseudo-code description of the employed algorithm is given in Algorithm 12. The low-level local search used in this algorithm uses the *1-shift* neighborhood in which one facility is reassigned to a different location than to which it is currently assigned. This results in a neighborhood size of $N \cdot (M-1)$ when it is exhaustively enumerated. The resulting locally optimal solution with respect to this neighborhood is then called *1-opt*. The partial evaluation of a move can be more efficient than a full evaluation and can be performed in $O(N)$, while a full solution evaluation requires $O(N^2)$.

In this study, two instances of ILS are used and both use a greedy strategy for accepting a new solution as a base for performing the next perturbation. Only a strictly better new solution is accepted. The two instances differ in their sampling and perturbation strategy. One instance is parameterless and similar to the multi-start local search as introduced in the QAP experiments. It uses an unbiased random sampling of solutions in both `Sample` and `Perturb`, while the second instance uses the greedy randomized construction described in Mateus et al. [MRS11] in `Sample` and uses a probabilistic reassignment of on average 10% of the facilities in `Perturb`. The value of 10% was determined using irace 2.4 on several benchmark instances in 5,000 experiments.

**Late-acceptance Hill Climber**

Late-acceptance hill climber (LAHC) [BB17] is an algorithm that was introduced more recently. It shares similarities with simulated annealing in that it uses a probabilistic acceptance criterion instead of a deterministic one. However, the temperature parameter in simulated annealing that governed the acceptance is replaced with a history. As the value of the temperature is highly dependent on the domain of the fitness values this parameter is subject to instance specific tuning, normalizing the fitness function or using the fitness of some initial randomly drawn solutions to automatically set the value. LAHC would also fit into the ILS framework, however, it does not employ local search and thus is better seen as a category of its own. Two parameterless variants were introduced [BL17] and of those pLAHC-s was the one

**Algorithm 12** Iterated Local Search for GQAP

```
1: procedure ILS(↓ pertstr)
2:     sol ← Sample()
3:     (bestsol, bestfit) ← OneOptLocalsearch(sol)
4:     while not Terminate() do
5:         s ← Perturb(bestsol, pertstr)
6:         (sol, fit) ← OneOptLocalsearch(s)
7:         if fit < bestfit then
8:             bestfit ← fit
9:             bestsol ← sol
10:        end if
11:    end while
12:    return (bestsol, bestfit)
13: end procedure
```

used in this study and thus, no parameters were tuned for this algorithm. A pseudo-code description is given in Algorithm 13 which is based on previous works (cf. [BL17] Algorithm 2 on page 223), but adapted to the notions, style, and abstraction used in this thesis. The algorithm uses the `GenerateNMove` neighborhood, for which $N = 1$ was chosen and thus is identical to the *1-shift* neighborhood used in the low-level local search of ILS.

**Evolution Strategy**

Evolution strategies have been introduced in Section 2.2.4 on page 28. They comprise a set of algorithms well-known for finding optima in real-valued search spaces. ES describe an adaptation of the perturbation / mutation strength based on the dynamics of the search. It uses a population of solutions and typically generates multiple offspring of which either an elite set is maintained together with the old population (termed "plus-ES") or an elite set is selected from the new population only (termed "comma-ES").

In this case study we implemented a (10, 1000)-ES ("comma variant") with recombination and mutation using the perturbation method that is also employed in ILS. However, instead of choosing a fixed strategy parameter of 10% that was tuned offline for ILS, we use a dynamic adaptation of the strategy parameter. This parameter governs how many of the facilities should be relocated. The value is transformed by a sigmoid function into the open interval $(0, 1)$ which is interpreted as a probability. The strategy parameter is mutated additively as it spans both negative and positive values including zero. The more conventional multiplicative approach is thus not applicable.

**Algorithm 13** Late Acceptance Hill Climber for GQAP, adapted from [BL17]

```
 1: procedure LAHC()
 2:     (bestsol, bestfit) ← Sample()
 3:     bestlist ← [bestfit]
 4:     for exp in 1..24 do
 5:         memory[1..2^exp] ← Initialize(bestlist)          ▷ Sorted afterwards
 6:         sprint ← 0
 7:         (sol, fit) ← (bestsol, bestfit)
 8:         while not Terminate() do
 9:             (move, movefit) ← GenerateNMove(sol)
10:             v ← sprint  mod |memory|
11:             if fit < memory[v] or fit ≤ movefit then
12:                 sol ← Apply(move, sol)                   ▷ Accept move
13:                 fit ← movefit
14:             end if
15:             if fit < memory[v] then
16:                 memory[v] ← fit                          ▷ Update memory
17:             end if
18:             sprint ← sprint + 1
19:             if fit < min(bestlist) then
20:                 bestlist ←+ fit                          ▷ Append to best list
21:                 bestfit ← fit
22:                 bestsol ← sol
23:             end if
24:         end while
25:     end for
26:     return (bestsol, bestfit)
27: end procedure
```

**Algorithm 14** Evolution Strategy for GQAP

```
1: procedure ES(↓ recombineyn, ↓ plusyn, ↓ μ, ↓ λ)
2:     (pop, σ) ← Sample(μ)                ▷ initialize population and strategy parameters
3:     while not Terminate() do
4:         nextgen ← [ ]
5:         for i in 1..λ do
6:             if recombineyn then
7:                 offspring ← DLX(pop)
8:                 σ′ ← Average(σ)
9:             else
10:                (offspring, σ′) ← Select(pop, σ)          ▷ Randomly select a parent
11:            end if
12:            σ″ ← σ′ + √2^−1 · 𝒩(0, 1)                  ▷ Mutate strategy parameters
13:            offspring′ ← Mutate(offspring, σ″)
14:            nextgen ←+ (offspring′, σ″)
15:        end for
16:        if plusyn then                ▷ In the plus-ES the old population is included
17:            nextgen ←+ (pop, σ)
18:        end if
19:        (pop, σ) ← Best(nextgen, μ)
20:    end while
21:    return BestOf(pop)
22: end procedure
```

The parameters of ES have been determined by irace 2.4 on several benchmark instances in 5,000 experiments. A pseudo code description of the implemented algorithm is given in Algorithm 14.

### Iterated Genetic Algorithm

Genetic algorithms have been introduced in Section 2.2.4 on page 26. We chose a more modern variant of genetic algorithms called *offspring selection genetic algorithms* which are described in more detail on page 27. The algorithm includes a success-based termination criterion. It stops when generating successful offspring in a certain generation requires too much effort. Very often it has been observed that this coincides with the convergence of the population and the loss of diversity [AWWB09]. Because of this termination procedure the algorithm cannot be run arbitrarily long, respectively it would not be meaningful to extend the algorithm past the point of convergence. Thus, we added a restart procedure where the population is reinitialized randomly, but not allowed to keep the same assignment that it had just converged to.

**Algorithm 15** Iterated Genetic Algorithm for GQAP

```
1: procedure OSGA(↓ popsize, ↓ mutprob)
2:     pop ← Sample(popsize)                          ▷ Initialize the population
3:     best ← BestOf(pop)
4:     while not Terminate() do
5:         nextgen ← [ ]
6:         restart ← false
7:         selpress ← 0
8:         while selpress ≤ 1 ∨ length(nextgen)/popsize ≥ selpress/500 do
9:             parents ← RandomSelect(pop)            ▷ Sample 2 solutions randomly
10:            offspring ← DLX(parents)
11:            Mutate(offspring, mutprob)
12:            if offspring betterThan BestOf(parents) then
13:                nextgen ←+ offspring
14:                best ← Update(best, offspring)
15:            end if
16:            selpress ← selpress + 1.0/popsize
17:        end while
18:        if length(nextgen) < popsize then          ▷ Population converged
19:            restart ← true
20:            nextgen ←+ BestOf(pop)                 ▷ Fill next generation with best of pop
21:        end if
22:        pop ← nextgen
23:        if restart then
24:            Mutate(pop)   ▷ Perform a relocation of all facilities to a different location
25:        end if
26:    end while
27:    return best
28: end procedure
```

For crossover a new operator was introduced by the author [BWA18]. Mutation is similar to ILS and applied to about 5% of the solutions, but with a higher strength (25%). The population size was chosen 500 which is a recommended default parameter [AWWB09] and was observed to be well suited to solve certain problem instances to optimality. Parents were selected randomly without bias as the selective pressure is put on offspring survival.

### Age-Layered Population Structure

The age-layered population structure (ALPS) is a complex evolutionary algorithm that also attempts to mitigate premature convergence, but in a different way than OSGA. It uses a number of stacked layers that contain

better and better solutions. The lowest of the layers is replaced with randomly generated solutions. In each generation mating pools are created for each layer together with the immediate lower layer. Each solution is attributed an age property in addition to its fitness. The age determines the layer that the solution is assigned to. Thus, as solutions participate in the variational process they become older and move from lower to higher layers. The variational process itself is identical to a standard genetic algorithm involving parental selection, crossover, and mutation.

ALPS was configured to use the same crossover and mutation as OSGA. It uses a maximum of $L = 10$ layers, and $N = 100$ solutions per layer. The age gap was set to $A = 20$ and a polynomial ageing scheme is employed. Further parameters were set to 1-elitism, and a generalized linear rank selection [TS95] with a selection pressure parameter set to 4. Drawing from previous experience and preliminary tests, this rather high value has proven to work successfully in many problems. A pseudo-code description of ALPS is given in Algorithm 16 and has in this form not been publicly made available elsewhere. Only the steady-state ALPS variant has published pseudo-code descriptions, but this variant has not been implemented.

**Greedy Randomized Adaptive Search Procedure**

The variant of GRASP used in this study is the algorithm described by Mateus et al. [MRS11] that was specifically implemented, analyzed, and tuned to optimize GQAP intances. We thus also follow their recommendations on parameters which have been extensively analyzed. A pseudo-code description is given in this thesis nevertheless in Algorithm 17, but which closely follows the original description and is rather adapted to the style and formatting in this thesis.

Because GRAPS is rather general its search performance is highly dependent upon a well-defined set of operators. These have been implemented exactly as described [MRS11] and their pseudo-codes are not repeated in this thesis, but are described in detail in the original publication. These operators are fairly complex and thus this GRASP algorithm is certainly a highly-tuned algorithm for GQAP. In contrast when we compare this GRASP algorithm with the ILS, the latter is much simpler and shorter to describe. Even the local search procedure employed as part of GRASP is of a special kind [MRS11].

As has been mentioned, we follow the authors recommendation of parameters that have been analyzed extensively and are as follows: the employed

**Algorithm 16** Age-Layered Population Structure for GQAP

```
1: procedure ALPS(↓ L, ↓ N, ↓ A, ↓ mutprob, ↓ selpress)
2:     layers_0 ← [ ]                                          ▷ An empty dummy layer
3:     gen ← 0
4:     A_i ← A · j   ∀(i, j) ∈ {(1,1), (2,2), (3,2^2), ..., (L-1, [L − 2]^2), (L, ∞)}   ▷ Age limits
5:     L ← 1                                                   ▷ Number of / last active layers
6:     while not Terminate() do
7:         if gen mod A = 0 then
8:             layers_1 ← Sample(N)                            ▷ (Re)Initialize the first layer
9:             sol_age ← 0   ∀sol ∈ layers_1                   ▷ Age is a property of solutions
10:        end if
11:        for i in 1..L do                                    ▷ Mating loop for layer i
12:            nextgen ← [ ]
13:            while Incomplete(nextgen) do
14:                (p′, p″) ← GeneralizedRankSelection((layers_{i−1}, layers_i), selpress)
15:                offspring ← DLX(p′, p″)
16:                Mutate(offspring, mutprob)                  ▷ Facility relocation
17:                offspring_age ← max(p′_age, p″_age) + 1
18:                nextgen ←+ offspring
19:            end while
20:            layers_i ← Replace(nextgen, layers_i)
21:        end for                                             ▷ End of mating loop for layer i
22:        MigrateElders(layers_{i−1}, layers_i, A_i) ∀i ∈ [2..L]
23:        if gen ≥ A_L ∧ L < L then
24:            L ← L + 1
25:            layers_L ← MatingLoop(layers_{L−1})   ▷ Copy layer and iterate one generation
26:        end if
27:        gen ← gen + 1
28:        best ← BestOf(best, ⋃_{i=1}^{L} layers_i)
29:    end while
30:    return best
31: end procedure
```

variant is termed "f-r-g-g", candidate size factor ($\eta$) = 50%, elite set size = 10, maximum candidate list size (mcl) = 10, $\delta = 4$, minimum elite set size ($\rho$) = 2, one-move probability = 50%, and maximum iterations for local search = 100 were set.

### Linearized Integer Programming

Two linearized integer programming (IP) models have been implemented that are described in the literature for the quadratic assignment problem (QAP) and which have been adapted to the GQAP. On the one hand there

**Algorithm 17** Greedy-Randomized Adaptive Search Procedure for GQAP, adapted from [MRS11]

```
1: procedure GRASP(↓ elitesetsize, ↓ ρ, ↓ δ, ↓ mcl, ↓ 1moveprob, ↓ maxiterLs, ↓ η)
2:     pop ← [ ]
3:     while not Terminate() do
4:         newbest ← false
5:         sol ← GreedyConstruction()
6:         if length(pop) ≥ ρ then
7:             if not IsFeasible(sol) then
8:                 sol ← SelectRandom(pop)
9:             end if
10:            ApproxLocalSearch(sol, mcl, 1moveprob, maxiterLs)
11:            other ← SelectRandom(pop)
12:            sol ← PathRelinking(sol, other, η)
13:            ApproxLocalSearch(sol, mcl, 1moveprob, maxiterLs)
14:            if sol betterThan best then
15:                best ← sol
16:                newbest ← true
17:            end if
18:            if length(pop) = elitesetsize then
19:                if newbest ∨ MaxSimilarity(sol, pop) ≥ δ then
20:                    candidates ← {p ∈ pop | p worseOrEqualThan sol}
21:                    if candidates ≠ ∅ then
22:                        pop ←⁻ arg max_{c ∈ candidates} Similarity(sol, c)
23:                        pop ←⁺ sol
24:                    end if
25:                end if
26:            else if newbest ∨ MaxSimilarity(sol, pop) ≥ δ then
27:                pop ←⁺ sol
28:            end if
29:        else if IsFeasible(sol) ∧ MaxSimilarity(sol, pop) ≥ δ then
30:            pop ←⁺ sol
31:        else if length(pop) = 0 then
32:            best ← BestOf(best, sol)
33:        end if
34:    end while
35:    return best
36: end procedure
```

is the linearization described by Frieze and Yadegar [FY83] and on the other hand the linearization described by Kaufman and Broeckx [KB78]. The models are implemented in the optimization programming language (OPL) and solved by IBM CPLEX[8] 12.7.0 for which an academic license had been obtained. IBM CPLEX is a commercial product that is widely used to solve linear mathematical programming formulations. The implementation of the solver is a secret however. The CPLEX solver was run with all parameters set to their default value. It does however feature many parameters that could be tuned. Certainly, we do not imply that the results show the performance of CPLEX as a whole, but rather of the default parameterization. The models are given in Listing 1 and 2. These models start by introducing the constants where the ellipsis (...) indicate that the data is defined in another, so called DAT-file. Then the decision variables are defined, followed by the objective and finally the constraints wrapped in the *subject to* block.

[8] `https://www.ibm.com/products/ilog-cplex-optimization-studio`

Listing 1: CPLEX-FY, an OPL implementation of the GQAP using Frieze and Yadegar linearization

```
int EQUIPMENTS = ...;
int LOCATIONS = ...;
float TC = ...;

range N = 1..EQUIPMENTS;
range M = 1..LOCATIONS;

float weights[N][N] = ...;
float distances[M][M] = ...;
float install[N][M] = ...;
float demands[N] = ...;
float capacities[M] = ...;

dvar int+ x[N][M] in 0..1;
dvar float+ z[N][M][N][M] in 0..1;

dexpr float installCosts = sum(i in N, k in M) x[i][k] * install[i][k];
dexpr float flowCosts = TC * sum(i in 1..EQUIPMENTS-1, k in M, j in N, h
    in M: j > i)
    (weights[i][j] * distances[k][h] + weights[j][i] * distances[h][k]) *
        z[i][k][j][h];

minimize installCosts + flowCosts;

subject to {

forall (i in N)
  AllAssigned:
  sum(k in M) x[i][k] == 1;

forall (k in M)
  Capacity:
  sum(i in N) x[i][k] * demands[i] <= capacities[k];

forall (i in 1..EQUIPMENTS-1, k in M, j in N: j > i)
  sum(h in M) z[i][k][j][h] == x[i][k];

forall (i in 1..EQUIPMENTS-1, j in N, h in M: j > i)
  sum(k in M) z[i][k][j][h] == x[j][h];

}
```

Listing 2: CPLEX-KB, an OPL implementation of the GQAP using Kaufman and Broeckx linearization

```
int EQUIPMENTS = ...;
int LOCATIONS = ...;
float TC = ...;

range N = 1..EQUIPMENTS;
range M = 1..LOCATIONS;

float weights[N][N] = ...;
float distances[M][M] = ...;
float install[N][M] = ...;
float demands[N] = ...;
float capacities[M] = ...;

float v[i in N][k in M] = sum(j in N, h in M) (weights[i][j] * distances[k
    ][h]);

dvar int+ x[N][M] in 0..1;
dvar float+ y[N][M];

dexpr float installCosts = sum(i in N, k in M) x[i][k] * install[i][k];
dexpr float flowCosts = TC * sum(i in N, k in M) y[i][k];

minimize installCosts + flowCosts;

subject to {

forall (i in N)
  AllAssigned:
  sum(k in M) x[i][k] == 1;

forall (k in M)
  Capacity:
  sum(i in N) x[i][k] * demands[i] <= capacities[k];

forall (i in N, k in M)
  v[i][k] * ( x[i][k] - 1 ) + sum(j in N, h in M) (x[j][h] * weights[i][j]
       * distances[k][h]) <= y[i][k];

}
```

### Hybrid Mathematical Programming Solver

To contrast the instances with another commercial product, LocalSolver 7.5 was included in the benchmark. LocalSolver[9] is a commercial mathematical programming solver [BEG+11]. Academic licenses can be obtained free of charge and are valid for one month. Models can be described in a similar way to OPL, but LocalSolver uses heuristics to solve them. Due to the heuristic

[9] http://www.localsolver.com

solver it is allowed to use non-linear functions and operators in the model and thus it is not necessary to perform a linearization. Two models have been implemented, the first model uses binary decision variables $x_{ij} \in \{0, 1\}$ to denote the assignment of facility $i$ to location $j$ while the second model uses integer decision variables $x_i \in M$ that directly encode the location to which facility $i$ is assigned to. An implementation of these models in the C# programming language and using the respective API is given in Listings 3 and 4. As in the case of CPLEX we used only default parameters and did not perform an extensive parameterization study.

Listing 3: C# implementation of the LocalSolver N model

```
LSModel model = new LocalSolver().GetModel();
// x[f] = l => equipment f is on location l
var x = new LSExpression[demands.Length];
for (int f = 0; f < demands.Length; f++)
  x[f] = model.Int(0, capacities.Length - 1);
// All locations contain not more equipments than there is capacity for
for (int l = 0; l < capacities.Length; l++) {
  var util = model.Sum();
  for (var f = 0; f < demands.Length; f++)
    util.AddOperand((x[f] == l) * demands[f]);
  model.Constraint(util <= capacities[l]);
}
// Create distances as an array to be accessed by an at operator
var distancesJagged = new double[capacities.Length][];
for (var i = 0; i < capacities.Length; i++) {
  distancesJagged[i] = new double[capacities.Length];
  for (var j = 0; j < capacities.Length; j++)
    distancesJagged[i][j] = distances[i, j];
}
var installJagged = new double[demands.Length][];
for (var i = 0; i < demands.Length; i++) {
  installJagged[i] = new double[capacities.Length];
  for (var j = 0; j < capacities.Length; j++)
    installJagged[i][j] = installationCosts[i, j];
}
LSExpression distancesArray = model.Array(distancesJagged);
LSExpression installCostsArray = model.Array(installJagged);
// Minimize the sum of product distance*flow
obj = model.Sum();
for (int f1 = 0; f1 < demands.Length; f1++) {
  for (int f2 = 0; f2 < demands.Length; f2++)
    obj.AddOperand(transportationCosts * weights[f1, f2] * distancesArray[
        x[f1], x[f2]]);
  obj.AddOperand(installCostsArray[f1, x[f1]]);
}
model.Minimize(obj);
```

Listing 4: C# implementation of the LocalSolver 01 model

```
LSModel model = new LocalSolver().GetModel();
// x[f,l] = 1 if equipment f is on location l, 0 otherwise
var x = new LSExpression[demands.Length][];
for (int f = 0; f < demands.Length; f++) {
  x[f] = new LSExpression[capacities.Length];
  for (int l = 0; l < capacities.Length; l++) x[f][l] = model.Bool();
}
// All equipments are installed in exactly 1 location
for (int f = 0; f < demands.Length; f++) {
  LSExpression nbLocationsAssigned = model.Sum();
  for (int l = 0; l < capacities.Length; l++)
    nbLocationsAssigned.AddOperand(x[f][l]);
  model.Constraint(nbLocationsAssigned == 1);
}
// All locations contain not more equipments than there is capacity for
for (int l = 0; l < capacities.Length; l++) {
  LSExpression assignedDemand = model.Sum();
  for (int f = 0; f < demands.Length; f++)
    assignedDemand.AddOperand(x[f][l] * demands[f]);
  model.Constraint(assignedDemand <= capacities[l]);
}
// Index of the assigned location of equipment f
var equipmentsOnLocations = new LSExpression[demands.Length];
for (int f = 0; f < demands.Length; f++) {
  equipmentsOnLocations[f] = model.Sum();
  for (int l = 0; l < capacities.Length; l++)
    equipmentsOnLocations[f].AddOperand(l * x[f][l]);
}
// Create distances as an array to be accessed by an at operator
var distancesJagged = new double[capacities.Length][];
for (var i = 0; i < capacities.Length; i++) {
  distancesJagged[i] = new double[capacities.Length];
  for (var j = 0; j < capacities.Length; j++)
    distancesJagged[i][j] = distances[i, j];
}
var installJagged = new double[demands.Length][];
for (var i = 0; i < demands.Length; i++) {
  installJagged[i] = new double[capacities.Length];
  for (var j = 0; j < capacities.Length; j++)
    installJagged[i][j] = installationCosts[i, j];
}
LSExpression distancesArray = model.Array(distancesJagged);
LSExpression installCostsArray = model.Array(installJagged);
// Minimize the sum of product distance*flow
var obj = model.Sum();
for (int f1 = 0; f1 < demands.Length; f1++) {
  for (int f2 = 0; f2 < demands.Length; f2++)
    obj.AddOperand(transportationCosts * weights[f1, f2] * distancesArray[
        equipmentsOnLocations[f1], equipmentsOnLocations[f2]]);
  obj.AddOperand(installCostsArray[f1, equipmentsOnLocations[f1]]);
}
model.Minimize(obj);
```

### 4.2.5 Experiment Setup and Execution

In contrast to the case study on QAP that was performed using open-source implementations of solvers only, the case study on GQAP includes a much more heterogeneous set of algorithm instances. We decided to measure run-length not in terms of evaluations, but in terms of the elapsed wall clock time. A reasonable abstraction for commercial solvers is difficult to attain and even difficult to measure. What is an "evaluation" in CPLEX and how to track their numbers? The wall clock time is still a valid measure, but of course this means that we are actually testing the performance of the implementation. Considerations such as runtime complexities of operations on collections and language details suddenly become relevant details. We also decided to go with a sequential instead of a parallel implementation, respectively assuming the algorithms are run in a single CPU environment. All open source implementations were conducted in C# using the .NET Framework 4.5.

For a fair comparison we used a single machine were all tests were performed. It is a virtual machine using Intel Xeon E5-2660 CPUs with a total of 28 cores with 125Gb RAM. The 28 cores were used to parallelize the experiment instead of the individual algorithms. Only CPLEX was allowed to take advantage of all 28 cores as it was the only exact solver in the test and is less of a direct competitor, but a different (exact) approach to problem solving.

A total of 30 to 45 repetitions were performed for every algorithm / problem instance pair for all open source implementations, while the commercial solvers were just executed once. The maximum runtime was limited to 1 minute. This was enough to find optimal solutions for several instances. The commercial solvers were run such that only one algorithm / problem instance pair was executing concurrently. The experiment yielded more than 75,000 runs.

Some runs of the pLAHC-s instance terminated before the available time budget and had not reached the target fitness. We simulated a restart procedure for those runs in that we sampled from the runs that we had observed. This is a process similar to bootstrapping.

The performance data that we obtained tracked the best found quality over time. We store the monotonic convergence graph for each run. In the open source implementations such a variable was updated after the call to the evaluation function, but was not made during an operator, such as a low-level local search. For CPLEX we used the MIPInfoCallback and the IterationTicked callback was used for LocalSolver. The execution time has been measured using

the System.Diagnostics.*Stopwatch* class which allows high-precision measurements. The experiment data has been published and is available online[10].

### 4.2.6 Performance Measurement

We use two dimensions to evaluate the performance of the methods in this study. On the one hand we record the achieved solution quality and on the other hand the execution time as the elapsed wall clock time. The best algorithm would achieve the best quality in less time than any other, however this is difficult to achieve. In general, longer runs typically lead to better quality than shorter runs. In this study we identify a target solution quality that is within a few percent of the best attained quality and measure the expected runtime (ERT) required to achieve it.

As in the previous study, we use six performance classes and assign the class values using Ckmeans.1dp [WS11a], while we reserve class 6 for all algorithm instances that did not achieve the target at all. Class 1 thus contains the best performing algorithm instances. Again, we use the $\log_{10}(\texttt{ERT})$ for clustering in order to distinguish the instances by performance in orders of magnitude rather than based on the actual runtime in seconds.

### 4.2.7 Post-hoc Experiment Analysis

A total of 12 algorithm instances was applied to 189 problem instances. We use target qualities of 1% and 2% to the optimum or the best-found quality if an optimum was unknown. For the 2% target on average we can observe that there are 1.4 algorithm instances with rank 1, 1.5 instances that achieved rank 2, also 1.5 instances with a rank of 3, and 1.2 respectively 0.9 ranked 4 and 5. Most algorithm instances (5.5 on average) did not achieve the target quality and ranked $6^{th}$. In the last case study we have observed several algorithm instances achieved good solutions, while in this case the field of successful algorithm instances per problem instance is much narrower. We state the percentage of class rankings observed for each algorithm instance in Tables 18 and 19. Again, it has to be noted that rank 1 only means a certain algorithm instance was fastest, not that it was the only algorithm instance to achieve that quality. Any of the algorithm instances classified among ranks 1 to 5 have *a chance* to achieve the target, but took increasingly longer. This probability may still be rather low within the given time budget.

[10] `dev.heuristiclab.com/AdditionalMaterial#GECCO2018`

At a 2% target, there were 126 (66.7%) cases where only a single algorithm instance was in rank 1, while in 11 (5.8%) cases, all but one algorithm instance failed to succeed. The binary model solved by LocalSolver achieved rank 1 most often still the question has to be raised if it would also constitute the best choice overall. The three algorithm instances that are especially interesting from a robustness perspective are ILS, OSGA, and GRASP+PR which failed to achieve the target quality for only 10-17% of the problem instances, while all other algorithm instances failed at a rate of about a third to two thirds. ILS would be a good choice as it achieved the 2% target in 90% of the instances.

However, caution has to be taken that this result is not misinterpreted. Since the performance classes are created based on the $\log_{10}$(ERT) it would be wrong to conclude that ILS is successful *90% of the time* it is applied. The result says only that there is some probability $> 0$ that ILS is successful on *90% of the problem instances* and this probability becomes smaller and smaller the higher the rank. Nevertheless, from an implementation perspective ILS is also attractive. While GRASP is a highly specialized algorithm with many tuned operators that is described and analyzed in a 39 page journal article [MRS11], ILS is a very simple implementation with just a perturbation operator and a low-level local search. ILS does perform worse than GRASP, however it is certainly faster to develop if starting from scratch and failed least often in this benchmark setting. The only static hyperparameter (perturbation strength) is also easy to be tuned and performs better than the dynamic adaption that we devised for ES which did not satisfactorily work.

The results in Table 19 show the performances of algorithm instances for a 1% target. There were 145 (76.7%) cases where only a single algorithm instance was in rank 1, while in 27 (14.3%) cases, all but one algorithm instance failed to succeed. As can be observed, OSGA solved more instances than the binary LocalSolver model and also failed to achieve the target least often (17%). While ALPS and pLAHC-s ranked very low in terms of the number of instances where they achieved rank 1, they still failed to reach the target in only around 40% of the problem instances and thus would be ranked $4^{th}$ and $5^{th}$ if the goal was to avoid rank 6 first and then achieve rank 1 most often.

We analyze potential correlations between landscape features and algorithm performance, in addition to correlations among algorithms in Table 20. From the point of portfolio building, given two positively correlated algorithm instances it would probably suffice to have one in the portfolio. However, negatively correlated algorithm performance indicate that one algorithm is good

Table 18: Percentage of ranks that algorithms achieved (2% target). [BWA18]

| Alg.Inst | $1^{st}$ | $2^{nd}$ | $3^{rd}$ | $4^{th}$ | $5^{th}$ | $6^{th}$ |
|---|---|---|---|---|---|---|
| LocalSolver 01 | 40% | 11% | 12% | 6% | 2% | 31% |
| OSGA | 28% | 26% | 18% | 10% | 5% | 14% |
| GRASP+PR | 25% | 32% | 13% | 8% | 4% | 17% |
| ILS | 19% | 31% | 26% | 14% | 1% | 10% |
| LocalSolver N | 10% | 14% | 10% | 5% | 4% | 58% |
| CPLEX-KB | 7% | 7% | 8% | 7% | 4% | 67% |
| CPLEX-FY | 6% | 14% | 8% | 5% | 2% | 65% |
| MLS | 4% | 5% | 5% | 7% | 13% | 66% |
| pLAHC-s | 3% | 5% | 13% | 23% | 23% | 34% |
| ALPS | 1% | 4% | 24% | 22% | 17% | 32% |
| ES | 1% | 4% | 12% | 15% | 14% | 55% |
| RS | 0% | 0% | 0% | 0% | 0% | 100% |

Table 19: Percentage of ranks that algorithms achieved (1% target)

| Alg.Inst | $1^{st}$ | $2^{nd}$ | $3^{rd}$ | $4^{th}$ | $5^{th}$ | $6^{th}$ |
|---|---|---|---|---|---|---|
| OSGA | 35% | 18% | 14% | 11% | 4% | 17% |
| LocalSolver 01 | 25% | 8% | 9% | 6% | 3% | 49% |
| GRASP+PR | 22% | 27% | 19% | 5% | 4% | 23% |
| ILS | 12% | 23% | 17% | 10% | 2% | 36% |
| LocalSolver N | 7% | 11% | 6% | 5% | 2% | 68% |
| CPLEX-KB | 7% | 7% | 4% | 5% | 2% | 75% |
| CPLEX-FY | 6% | 14% | 7% | 6% | 1% | 67% |
| MLS | 3% | 3% | 4% | 5% | 13% | 71% |
| pLAHC-s | 3% | 3% | 11% | 20% | 22% | 42% |
| ALPS | 1% | 8% | 16% | 19% | 16% | 40% |
| ES | 0% | 3% | 5% | 8% | 8% | 76% |
| RS | 0% | 0% | 0% | 0% | 0% | 100% |

especially at those instances that another is bad at and vice versa. These algorithms are then highly interesting to form a portfolio with as their performances complement one another. For instance, the OSGA instance is such a candidate as it has significant correlations with ALPS ($\rho \approx 0.51, p < 1e^{-6}$), but performs better in general. On the other hand it is negatively correlated to MLS and pLAHC-s, albeit with lower significance ($\rho \approx -0.26, -0.34, p < 0.05, 1e^{-3}$). There is no significant correlation to other algorithm instances. The two CPLEX models that we employed in the test performed similarly ($\rho \approx 0.71, p < 1e^{-6}$), however the two LocalSolver models are not correlated. The LocalSolver model with only binary decision variables performs much better than the model with integer decision models from a greater domain. Both encodings are equally powerful and cover the same solution space. The difference in performance are most probably due to internals of LocalSolver, which are not in the public domain.

But we do not only look for correlations between solver performance. In Table 20 we also state correlations between landscape characteristics and algorithm ranks. Unexpectedly, we can observe high correlation between exact solvers and problem dimension ($\rho \approx (0.74, 0.57)$) while other algorithm instances are less sensitive to dimensionality. LocalSolver even achieved a slight negative correlation suggesting that it was performing better at larger instances (lower rank) than other algorithm instances. When examining correlations of $|M|/|N|$ and relative algorithm performance we see that most instances achieved worse performance when this ratio went up and thus when the amount of locations was closer to the amount of facilities - which also corresponds to an increase in solution space relative to $N$. However, pLAHC-s showed a slight negative correlation indicating that it performed slightly better in such cases. This raises a question for future studies if this may be rooted in a probabilistic acceptance criterion. The most interesting correlation among solver performance and problem specific landscape characteristic is with respect to utilization. We can observe that most algorithm instances show a positive correlation indicating that they achieved higher rank on problem instances with higher utilization. However, OSGA and ALPS, as representatives of genetic algorithm variants, show a negative correlation ($\rho \approx (-0.38, -0.3)$) and thus those two are more successful on those instances relative to the other algorithms. We hypothesize that crossover is beneficial to find good solutions for problem instances with high utilization, whereas it prevents good performance in cases with low utilization. Still, more research is required as ES was

also employed with crossover, but performed very poorly overall.

The commercial solvers CPLEX and LocalSolver were not affected much by utilization and showed the smallest absolute correlation. However, we can see that $W_0$ and also $D_0$ have a stronger influence, especially on CPLEX. Naturally, CPLEX benefits a lot from sparse matrices, because this reduces the problem size considerably as it eliminates a lot of terms in the objective. LocalSolver is mixed, while the integer model does benefit, solving the binary model does not profit from more sparse problem instances. It is also interesting to observe that the two LocalSolver models are likely solved much differently than CPLEX solves the two linearizations. Often the correlations have different sign for the binary and the integer model, while for CPLEX it's about the same.

An analysis of the performance with respect to ECDF curves as mentioned in Section 2.5 on page 50 is given in Figure 33 [BWA18]. We may see that ILS and GRASP are very good initially and both share the same greedy construction heuristic. But while GRASP manages to achieve the 2% target faster, ILS falls behind after about 1 second. Both are surpassed slightly before the 10 second mark by OSGA and later by LocalSolver which achieved most of the 2% targets. It is also necessary to observe that even for such short runtimes of one minute, a poor metaheuristic, as our ES implementation in this case, is outperformed by exact approaches. Of course, the comparison involves much more cores on the CPLEX side. Assuming a perfect discriminator, selecting from a portfolio consisting of OSGA, GRASP, and the LocalSolver 01 model a total of 74% of the problem instances can be efficiently solved to 2%, which increases to 83% when ILS is added.

In Figure 34 we observe a t-sne projection [MH08] of the problem instances in the study. The projection was performed using the *problem specific* features $|N|, |M|/|N|,$ `util`, and `feas` as described in Section 4.2.2. For each algorithm instance a separate plot is shown where the problem instances as points are colored according to the performance classes of that algorithm instance. In Figure 35 projections based on different feature sets are compared for the performance of the OSGA algorithm instance.

In Figure 36 we observe the performance of OSGA and GRASP on a parallel coordinates plot. The three different feature are shown on the x-axis and their respective standardizes values on the y-axis. It can be seen, that there is some difference, with respect to $|M|/|N|$ (M/N Ratio) and `bump` (Bumpiness). Problem instances with larger values of bumpiness are solved more efficiently by OSGA while problem instances with smaller bumpiness favor GRASP.

Table 20: Correlation between fitness landscape characteristics and algorithm instances' ranks using Spearman's $\rho$. The upper diagonal displays significant correlations similar to Table 17. [BWA18]

| | ILS | MLS | LAHC | ES | OSGA | ALPS | GRASP | CPLEX-FY | CPLEX-KB | LocalS. 01 | LocalS. N | RS | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ILS | 1.00 | | *** | | | | ** | | | | | | 0.18 |
| MLS | 0.15 | 1.00 | ** | *** | * | | *** | *** | *** | | *** | | 0.34 |
| LAHC | 0.50 | 0.37 | 1.00 | *** | ** | | *** | | | | | | 0.23 |
| ES | 0.19 | 0.49 | 0.40 | 1.00 | | * | *** | ** | *** | | *** | | 0.37 |
| OSGA | 0.01 | -0.26 | -0.34 | 0.01 | 1.00 | *** | | | | | | | 0.06 |
| ALPS | -0.04 | 0.12 | -0.15 | 0.25 | 0.51 | 1.00 | | ** | * | | * | | 0.24 |
| GRASP | 0.33 | 0.49 | 0.44 | 0.62 | 0.01 | 0.19 | 1.00 | ** | ** | | *** | | 0.36 |
| CPLEX-FY | -0.04 | 0.62 | 0.12 | 0.35 | -0.07 | 0.37 | 0.32 | 1.00 | *** | | ** | | 0.31 |
| CPLEX-KB | -0.01 | 0.63 | 0.19 | 0.48 | -0.06 | 0.30 | 0.38 | 0.71 | 1.00 | | *** | | 0.35 |
| LocalS. 01 | -0.04 | -0.01 | 0.05 | 0.20 | -0.12 | 0.07 | 0.17 | -0.09 | 0.04 | 1.00 | | | 0.11 |
| LocalS. N | 0.08 | 0.42 | 0.12 | 0.44 | 0.04 | 0.26 | 0.40 | 0.37 | 0.48 | 0.07 | 1.00 | | 0.31 |
| RS | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.08 |
| feas. | -0.14 | -0.38 | -0.14 | -0.36 | -0.19 | -0.34 | -0.42 | -0.2 | -0.37 | -0.09 | -0.33 | 0.00 | -0.25 |
| $CV(B)$ | -0.02 | 0.06 | 0.06 | 0.15 | 0.2 | 0.31 | 0.1 | 0 | 0.12 | 0.11 | 0.03 | 0.00 | 0.09 |
| $CV(R)$ | -0.03 | -0.01 | -0.08 | 0.09 | 0.1 | 0.17 | 0.08 | 0.03 | 0.02 | 0.06 | 0.06 | 0.00 | 0.04 |
| $\lvert N\rvert$ | -0.06 | 0.48 | 0.03 | 0.32 | 0.03 | 0.48 | 0.34 | 0.74 | 0.57 | -0.11 | 0.32 | 0.00 | 0.26 |
| $CV(D)$ | 0.1 | 0.22 | 0.04 | 0.17 | 0.13 | 0.35 | 0.17 | 0.4 | 0.28 | -0.11 | 0.27 | 0.00 | 0.17 |
| $D_0$ | 0.01 | -0.45 | 0.07 | -0.25 | -0.16 | -0.43 | -0.26 | -0.46 | -0.57 | 0.09 | -0.4 | 0.00 | -0.23 |
| $CV(W)$ | 0.07 | -0.02 | 0.1 | -0.14 | -0.1 | -0.09 | -0.1 | -0.02 | 0.11 | -0.1 | -0.01 | 0.00 | -0.03 |
| $W_0$ | 0.19 | -0.31 | 0.1 | -0.05 | 0.11 | -0.02 | -0.04 | -0.49 | -0.36 | 0.26 | -0.1 | 0.00 | -0.06 |
| $M/N$ | 0.02 | 0.31 | -0.18 | 0.26 | 0.3 | 0.31 | 0.2 | 0.23 | 0.36 | 0.03 | 0.4 | 0.00 | 0.19 |
| util. | 0.39 | 0.47 | 0.52 | 0.26 | -0.38 | -0.3 | 0.47 | 0.17 | 0.22 | -0.05 | 0.09 | 0.00 | 0.16 |
| bump. | 0.21 | 0 | 0.31 | -0.12 | -0.46 | -0.5 | 0.06 | -0.16 | -0.19 | -0.04 | -0.22 | 0.00 | -0.09 |
| flat. | -0.14 | 0.38 | -0.14 | 0.34 | 0.26 | 0.48 | 0.18 | 0.4 | 0.45 | -0.03 | 0.36 | 0.00 | 0.21 |
| sharp. | -0.03 | -0.41 | -0.1 | -0.31 | -0.13 | -0.26 | -0.29 | -0.39 | -0.34 | 0.09 | -0.29 | 0.00 | -0.21 |
| ac1 | 0.01 | 0.34 | 0.02 | 0.29 | 0.05 | 0.09 | 0.34 | 0.18 | 0.22 | 0 | 0.19 | 0.00 | 0.14 |
| c.len | 0.01 | 0.32 | 0 | 0.28 | 0.06 | 0.08 | 0.32 | 0.18 | 0.22 | 0.02 | 0.19 | 0.00 | 0.14 |
| dbi | -0.33 | -0.41 | -0.36 | -0.08 | 0.35 | 0.32 | -0.27 | -0.11 | -0.12 | 0.05 | -0.09 | 0.00 | -0.09 |
| div. | -0.05 | -0.15 | -0.06 | -0.1 | 0.08 | 0.23 | -0.03 | 0.17 | -0.02 | -0.04 | -0.07 | 0.00 | 0.00 |
| ic | -0.29 | -0.46 | -0.32 | -0.25 | 0.14 | -0.12 | -0.51 | -0.41 | -0.35 | 0.04 | -0.26 | 0.00 | -0.23 |
| is | -0.32 | -0.52 | -0.34 | -0.36 | 0.08 | 0.04 | -0.51 | -0.17 | -0.29 | 0.02 | -0.31 | 0.00 | -0.22 |
| pic | 0.36 | 0.38 | 0.35 | 0.13 | -0.23 | -0.11 | 0.43 | 0.25 | 0.2 | -0.06 | 0.16 | 0.00 | 0.16 |
| dbi* | -0.33 | -0.35 | -0.35 | -0.11 | 0.27 | 0.28 | -0.28 | -0.06 | -0.09 | 0.07 | -0.1 | 0.00 | -0.09 |
| ic* | 0.32 | 0.32 | 0.36 | 0.1 | -0.26 | -0.28 | 0.28 | 0.05 | 0.08 | -0.06 | 0.1 | 0.00 | 0.08 |
| reg. | -0.08 | -0.27 | -0.11 | -0.19 | 0.05 | 0.13 | -0.14 | 0.06 | -0.14 | -0.05 | -0.17 | 0.00 | -0.08 |
| $H(X)$ | -0.3 | -0.46 | -0.33 | -0.25 | 0.15 | -0.1 | -0.51 | -0.4 | -0.35 | 0.04 | -0.26 | 0.00 | -0.23 |
| Avg.(Abs) | 0.17 | 0.33 | 0.23 | 0.26 | 0.19 | 0.25 | 0.30 | 0.27 | 0.29 | 0.10 | 0.24 | 0.03 | |

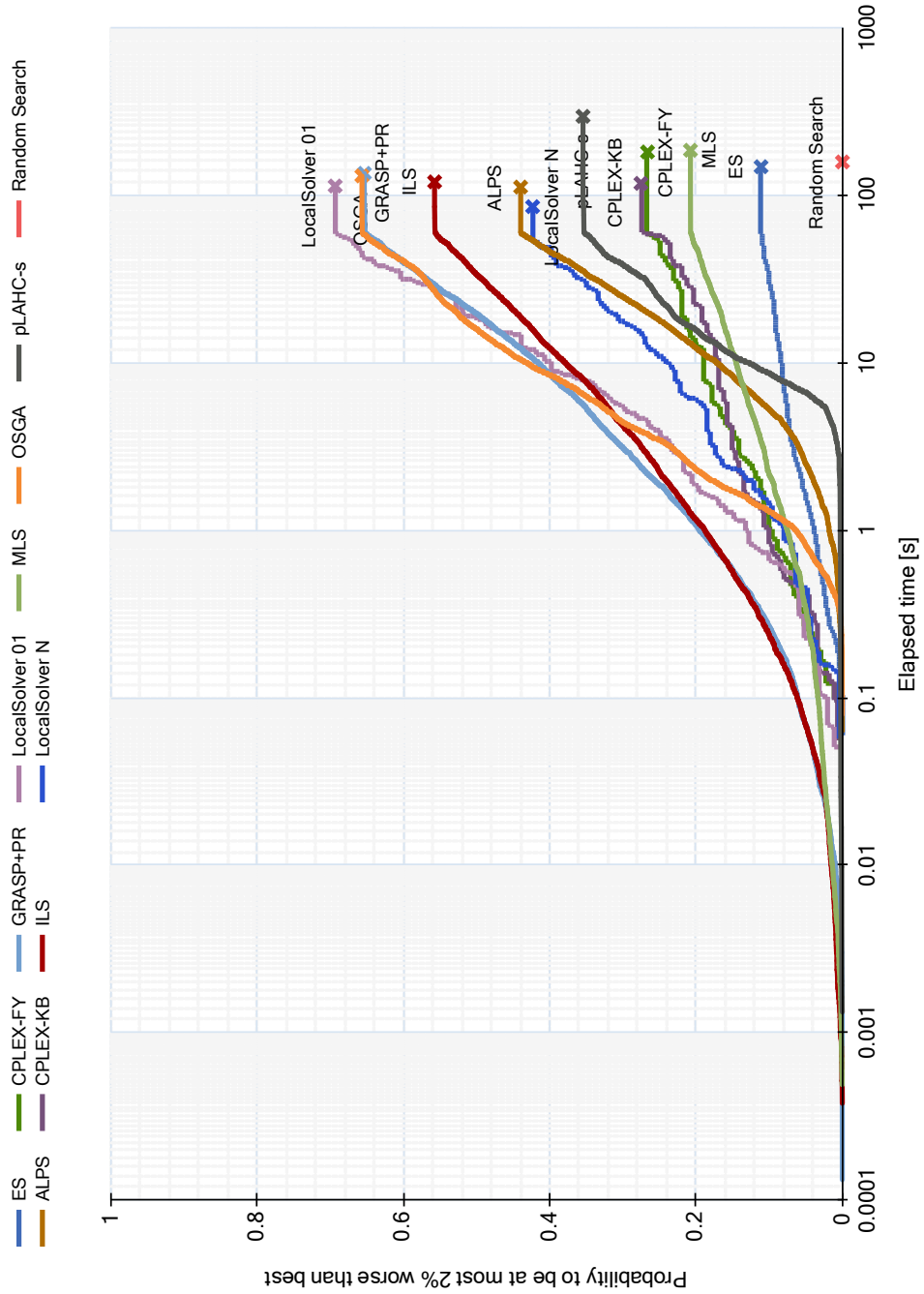



Figure 33: Runtime analysis of different algorithm instances for the 2% target on the GQAP benchmark problem set.

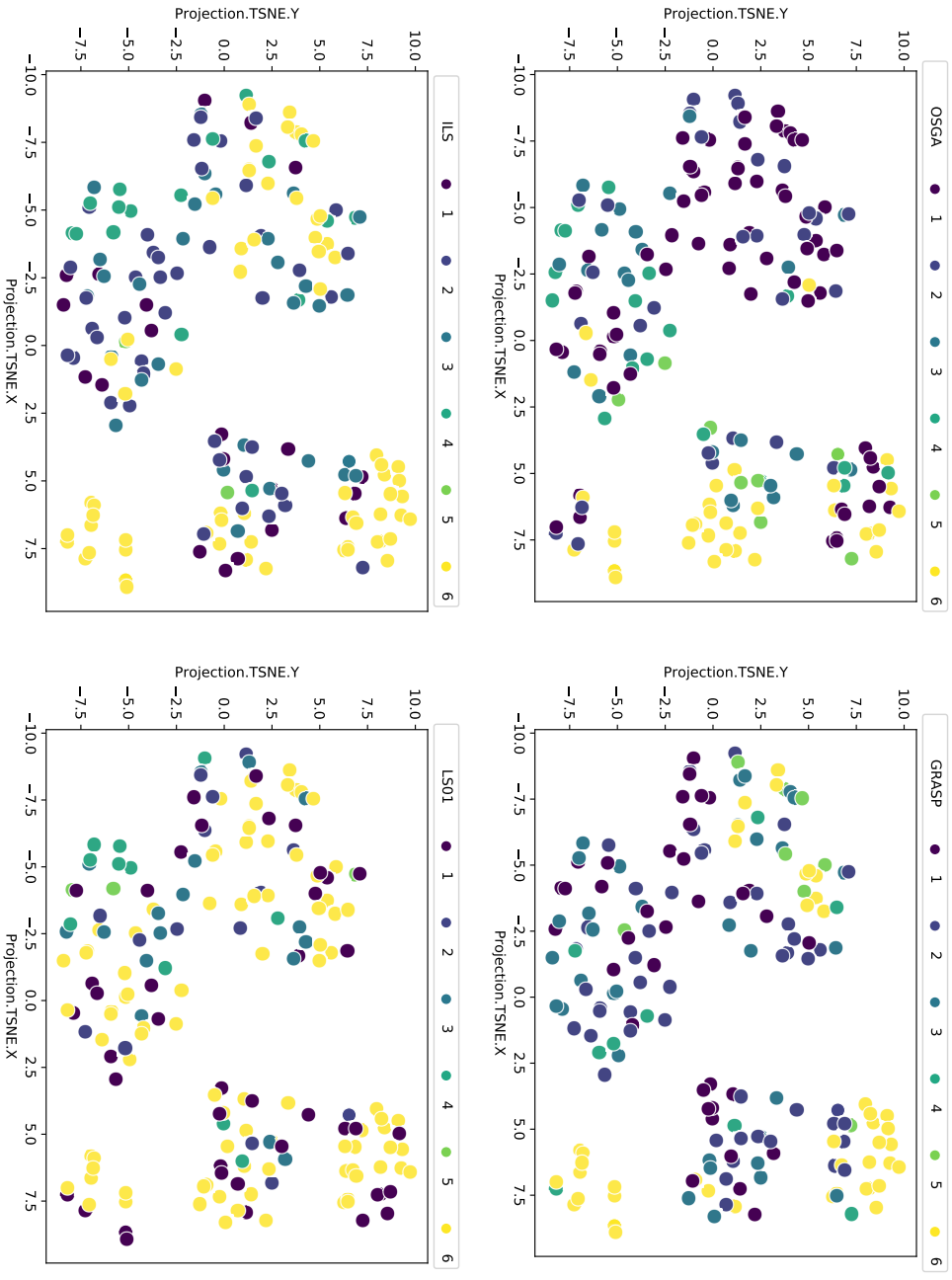



Figure 34: Performance classes for the 1% target for OSGA, GRASP, ILS, and LocalSolver 01 (left to right, top to bottom). Each dot represents a problem instance projected from problem specific feature space with t-sne.

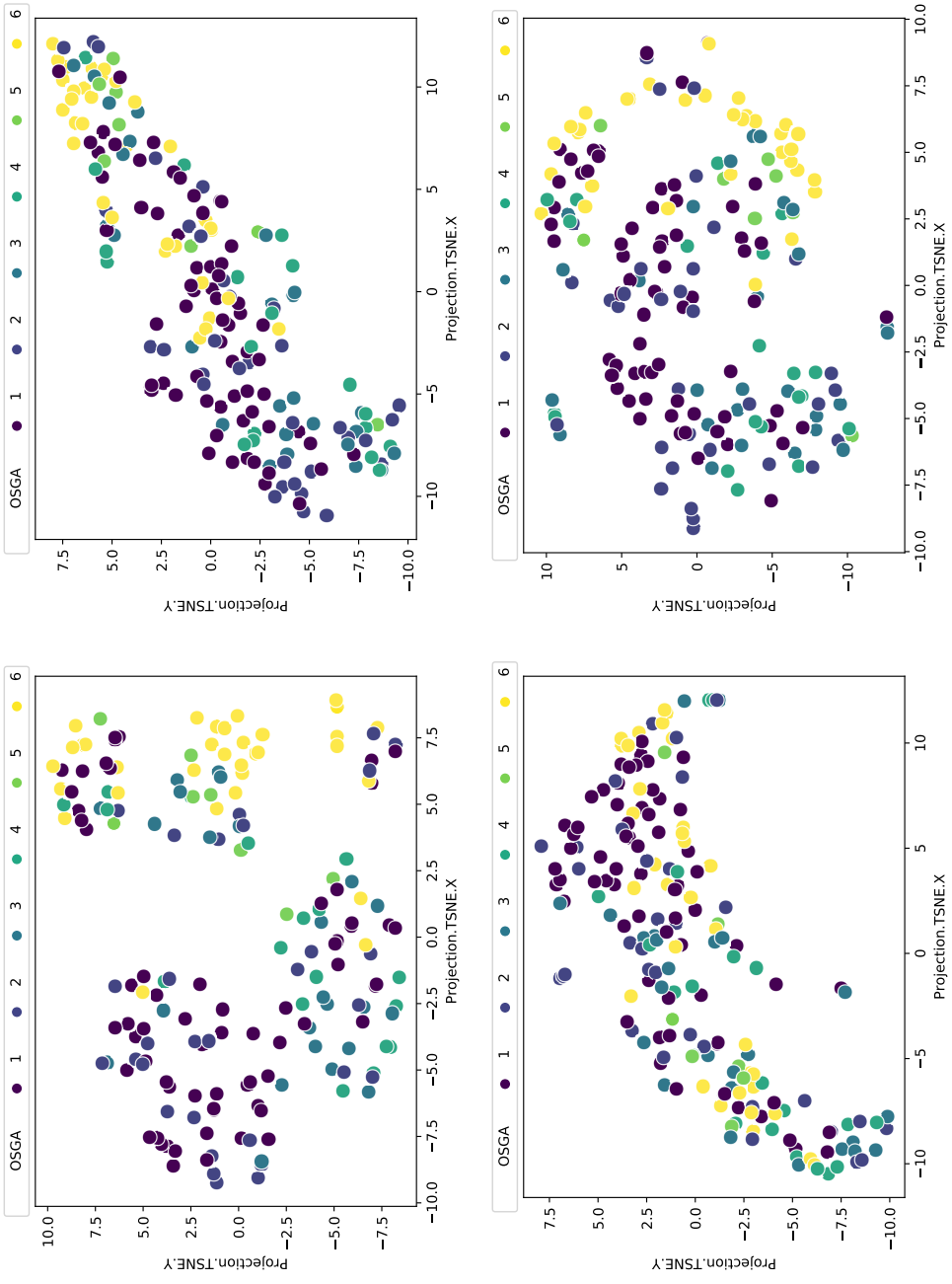



Figure 35: Performance classes for the 1% target for OSGA only, but using different features for performing the projection: problem specific, directed walk, random walk, and all combined (left to right, top to bottom). Each dot represents a problem instance projected with t-sne.

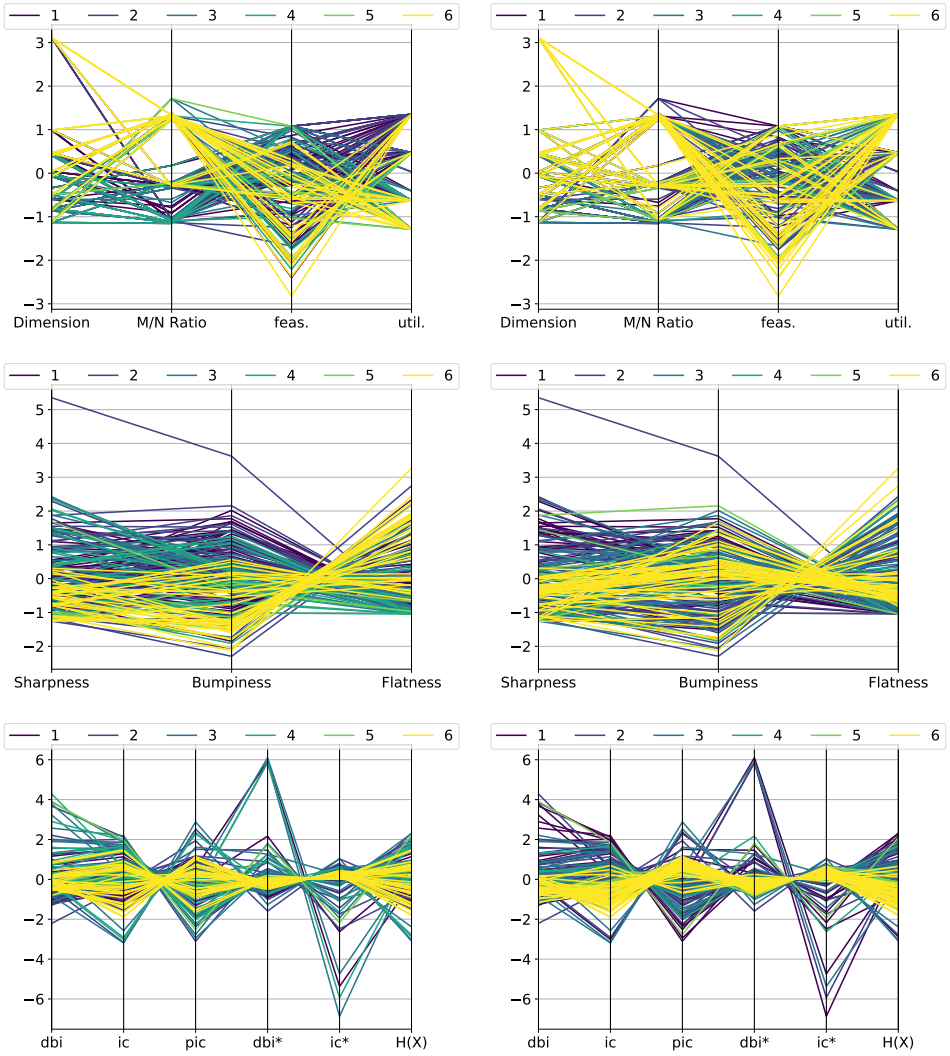



Figure 36: Parallel coordinates plot for the 1% target for OSGA (left) and GRASP (right) of different features (top to bottom): problem specific, directed walk, random walk. The color represents the performance class with class 6 in light yellow represents very bad performance and darker blue represents very good performance.

### 4.2.8 Algorithm Selection

In this section we will use a k-nearest neighbor algorithm (k-NN) to perform the algorithm selection and evaluate its performance. k-NN is an instance-based machine learning method that does not learn a model. Its performance depends on the hyperparameter $k$ and the distance metric used in the feature space. In this case we will stick with a Euclidean distance (L2-norm), but vary the feature sets which have a strong influence on the distance as we have seen in Figure 35. The feature sets that we will compare are problem specific features, random walk-based features, directed walk-based features, and all features combined.

The ranking of algorithm instances is composed of the observed performances of the nearest $k$ problem instances. For $k = 1$ the performance is exactly the same, but for $k > 1$ an averaging must be used. It has however been hypothesized that such an averaging does not favor the best, but rather a generally well performing method [BAW17]. Thus we use $k = 1$ for all experiments and evaluate its performance. Again, a 2% target has been used to the optimum respectively the best found solution. All features are normalized to 0 mean and unit variance before the L2-norm is used to compute the distance. A leave-one-out crossvalidation (LOOCV) is used to evaluate the generalization performance. Thus, a total of 189 folds are evaluated and a ranking should be achieved. Similar to the previous study we use Spearman's $\rho$ and NDCG to evaluate the performance of the recommender.

### 4.2.9 Results

If all characteristics are used the combined solver achieves better performance than any of the individual solvers as shown in Table 21. It achieved a rank 1 algorithm instance in 41% of the cases, and an unsuitable instance in only 17% of the cases. Thus algorithm selection results in a better overall solver in that it aims to pick the better ones for a concrete problem instance.

Furthermore, we analyze the $\text{NDCG}_1$ performance and the correlation coefficient in Table 22. The best results are achieved with the problem specific features only. Neither the random walk nor the directed walk exploratory landscape analysis results in an improvement of the recommendation.

In this experiment all 12 algorithm instances are used, but for future work it could be meaningful to limit the analysis to only the top four algorithms which may account for a large number of problem instances.

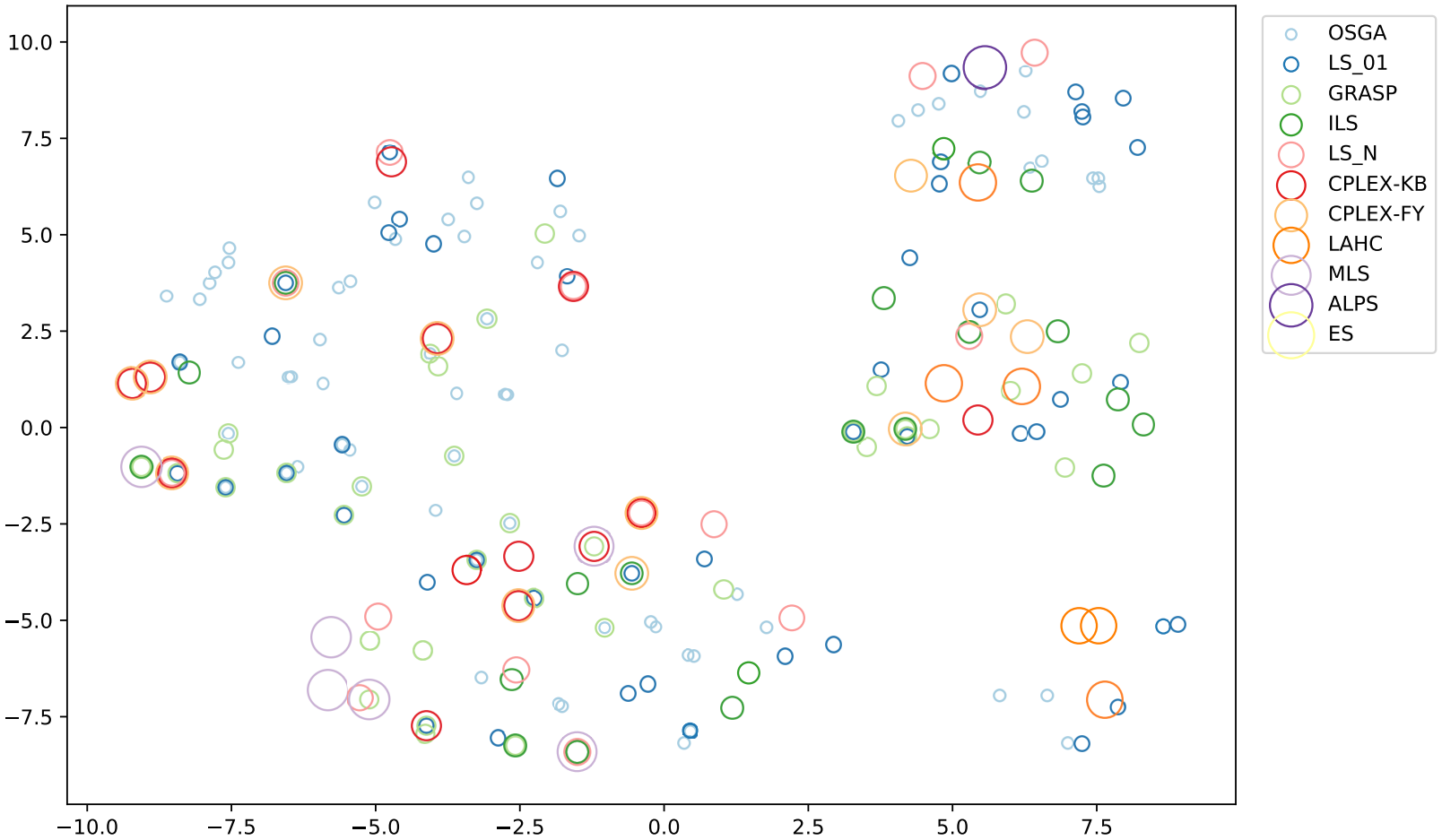



Figure 37: Problem instance map showing the best performing algorithm instance(s) for the 1% target using problem specific features for performing the projection. Each dot represents a problem instance projected onto a plane with t-sne.

Table 21: The combined solver given the one nearest neighbor achieved better performance than the individual solvers for the 2% target.

| Alg.Inst | $1^{st}$ | $2^{nd}$ | $3^{rd}$ | $4^{th}$ | $5^{th}$ | $6^{th}$ |
|---|---|---|---|---|---|---|
| **Combined** | 41% | 22% | 12% | 5% | 3% | 17% |
| LocalSolver 01 | 40% | 11% | 12% | 6% | 2% | 31% |
| OSGA | 28% | 26% | 18% | 10% | 5% | 14% |
| GRASP+PR | 25% | 32% | 13% | 8% | 4% | 17% |
| ILS | 19% | 31% | 26% | 14% | 1% | 10% |

Table 22: Correlation analysis of an ideal selection and with k-NN [BWA18]

| Features | $NDCG_1$ | $\rho$ |
|---|---|---|
| problem specific | 0.69 | 0.49 |
| random walk (fla) | 0.56 | 0.39 |
| directed walk (fla) | 0.65 | 0.43 |
| problem specific + fla | 0.68 | 0.48 |
| problem specific + directed walk | 0.69 | 0.49 |

As a last result, we developed a new visualization that has already been shown in Figure 32 for the previous study. In this new visualization we want to show the best algorithm instances for each problem instance in one plot, rather than in many different plots. We have to overcome an obstacle in that multiple algorithm instances may share rank 1 for certain problem instances, thus we have to find a suitable way to display such an overlap. In Figure 37 we sorted the instances by the number of rank 1 instances and assign differently sized circles as markers. Each problem instance is then displayed as the corresponding circles for the respective algorithm instance that solved it most efficiently. Because, the circles are increasing in size and shown without a fill color, different algorithm instances achieving top performance stack together. Thus we can also easily visualize instances that are easy to solve in that there are more choices of efficient algorithms. Again, to identify the locations of the problem instances, we used t-sne as a projection of the high dimensional feature space into two dimensions.

### 4.2.10 Conclusion and Outlook

In this section we performed a landscape analysis on generalized quadratic assignment problems, we devised new highly efficient algorithms and models and applied them in a large study involving more than 70,000 experiments. We analyzed correlations among the performance of these algorithms and among performance and features. We also showed in more depth different visualizations that are based on projecting the problem instances, in this case with t-stochastic neighbor embedding (t-sne), which is a very well known and highly suited technique for this task. Furthermore, we evaluated the performance of a simple one nearest neighbor recommender based on various feature sets and showed that the resulting performance is better than that of an individual solver. In this study, we also evaluated a mix of commercial and open source solvers, and found both to be quite efficient. Certainly, CPLEX as the only exact solver, did have its limitations, however LocalSolver could provide very good solutions in short time.

We developed a new visualization for showing the performance of algorithm selection as a classification problem when there exist multiple algorithm instances for class 1. This visualization is based on a scatter plot with stacked markers. Additionally, we identified a new hypothesis regarding a relationship between the success of crossover and the utilization of the problem instance [BWA18]. This has to be investigated further in future work.

# 5 Decision Support Systems for Real-World Applications

> "Plans are worthless, but
> planning is everything"
>
> Dwight D. Eisenhower [Bla57]

Given the developments that we outlined so far in this thesis, we are closer to think of and create decision support systems in heuristic optimization. We do not only want to compare performances among algorithms, although that is a prime interest in scientific research, but also suggest algorithm instances to apply to previously unknown problem instances. In addition, some users of a possible decision support system, would certainly value the ability of searching the solution space not only by starting from randomly generated solutions, but by reusing previously identified solutions of higher quality. In addition, users may contribute to this system by supplying algorithm and problem instances, runs and solutions. But before talking about such a possible decision support system more in detail we have to reconsider the decisions that have to be made, the information that leads to well-funded decisions and their outcome and impact.

We will focus on the whole process of applying heuristic optimization from the problem definition to the presentation of a valid and acceptable solution. In general, we will assume the research model depicted in Figure 38 is followed which has been formulated by Sagasti and Mitroff [SM73, MBPS74] and has been discussed since [BF02] in the field of operations research (OR). In this model the real problem situation is transformed into a scientific model to which solutions are found and which are then implemented in the real world again. To arrive at a scientific model a conceptual model is first developed in a phase called "conceptualization". Sagasti and Mitroff put extra emphasis on the importance of the conceptual model, but found it difficult to formalize. They speculated that "perhaps this conceptualization process is a part of the *art* of operations research rather than the *science* of OR" [SM73]. Also it is important to mention that the solution may influence the conceptual model and vice versa in such a way that putting the scientific model to work actually produces usable solutions for the real world, or at least the conceptual model of it. Another important arrow is drawn between reality and scientific model emphasizing the validation process.

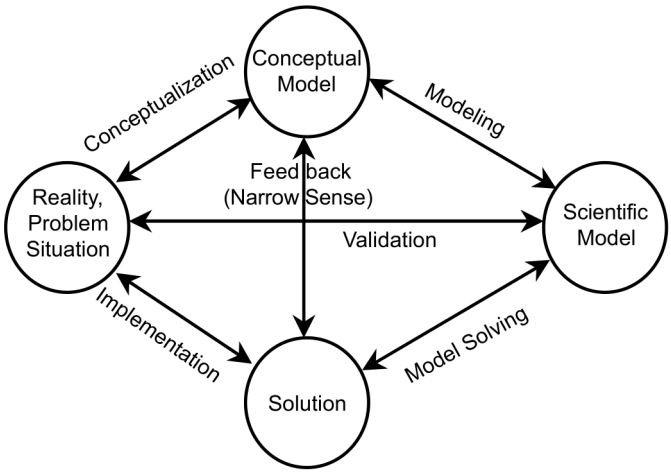



Figure 38: Research model as displayed in [BF02] originally published by [MBPS74].

This research model is highly useful in depicting the basic steps in problem solving. In each process certain decisions are made, however not all of them may be supported by systems. Nevertheless, in the following we will cover some important aspects and aim to motivate the use of decision support.

## 5.1 Motivation for Decision Support

In more recent years there have been continued research efforts towards methods that automatically select from a pool of algorithm instances suitable to solve a given problem instance [SM08, XHHLb08, XHLb10, SMvH11]. This is highly useful in the case of choosing the right solver for a given task, e.g. in an automated decision scenario. However, the idea of providing a decision support system (DSS) extends beyond this case. In the greater context of the research model (cf. Figure 38) this covers only the model solving processes that researchers encounter when tackling real-world problems. In this narrow focus, such a DSS should be able to help in the understanding of the problem landscape. Which instances are related to each other in terms of their characteristics? Furthermore, a DSS should support users in applying search algorithms, for instance by suggesting efficient solvers or by seeding solvers with some solutions in order to search more actively in a certain part of the search space. Finally, the results should be documented and intuitive comparisons between solvers should be possible.

## 5.2 Decision Support with the Research Model

A decision support system may become effective when it supports mastering a number of processes encountered in the general research model. Especially, concerning real-world problems and application scenarios, the conceptualization phase governs a few critical decisions. The objective has to be defined in terms of a measurable quantity that should either be minimized or maximized. Then it must be decided which kind of modeling methodology is suitable to achieve an improvement with respect to the objective. Finally, a solution approach must be designed and the resulting solutions must be screened and returned to the user.

**Goal**

Predating any formalized approach in minimizing or maximizing a certain target, the problem situation has to be analyzed. In industry projects practitioners often formulate only business goals which can be as coarse as "we want to improve utilization" or "we want to automatize crane operations". This part of the conceptualization process is difficult to support and requires a lot of experience also involving creativity, whole-system thinking, but also analytical abilities. Nevertheless, it is among the most important steps. An incomplete conceptual model that does not consider related activities or the wrong goal can lead to unforeseen consequences and unwanted side-effects in the end. Also, often there are several goals and also constraints: 'We have to improve X, but still do Y". At a moment where formalization is still low, it is uncertain if these goals are even feasible to achieve.

**Modeling Methodology**

Modeling methodology is an important decision which also has strong influence on how the model can be solved. A problem situation that has to be simulated may hardly be solved by exact methods and often requires a heuristic approach. On the other hand having a mathematical model, there may be highly efficient exact solvers. The options can be coarsely summarized as

- Mathematical Programs
- Deterministic Simulations
- Stochastic Simulations

Mathematical programs often provide a concise description of the optimization problem. However, large number of constraints or many decision variables

can lead to lengthy descriptions. Formulations might become more complicated due to linearization techniques or other relaxations that are performed for efficient solving. The quadratic assignment problem (QAP) formulation given in Equations (2.3) to (2.6) is an example of a mathematical program.

Deterministic simulations are similar to mathematical programs, but generally allow more complex descriptions of the problem. Any mathematical description can easily be transformed into a deterministic simulation. These often accept a complete solution of the problem and compute several characteristics, e.g. objective value and constraint violations. In the QAP case this would be the sum of the flow distance products between all assigned facilities. Even stochastic problems, such as the probabilistic traveling salesperson problem, may be represented as deterministic simulation or mathematical program. For instance, by deriving an analytic expression of the desired objective, e.g. expected distance or by sampling from the probabilities, i.e. creating realizations, and optimizing the expectation given those samples.

The domain of stochastic simulations is certainly the most challenging from the point of model solving. Each run of the simulation will likely result in a different outcome. Typically, this approach is referred to as simulation-based optimization, because it includes additional activities such as having to estimate the expected value from a range of samples. Comparing solutions to stochastic simulations is akin to the comparison of probabilistic variables and thus more complex than a simple $>$ or $<$ operation as in the case of deterministic simulations. Statistical tests may have to be performed to decide whether the means of two solutions are indeed different and if not additional simulations may have to be performed. Thus, often a number of repetitions may be necessary using random seeds or a fixed set of seeds. The expected value may then estimated as the average of the samples' outcomes. In the case of a fixed set of seeds this could again lead to a deterministic approach.

**Solution Representation**

The solution structure, i.e. the decision variables and their interrelations, need to be determined. The type of the variables, typically discrete or continuous numbers, need to be fixed. However, there can also be more complex solution types with a dynamic amount of decision variables such as whole computer programs. These may be used within a simulation, e.g. to perform a ranking of jobs such as in scheduling problems [BWWA08].

Not all algorithms are well defined for all possible solution representations.

In continuous optimization the notion of gradient and direction is present which is missing in e.g. permutation spaces. Thus, some algorithms that may approximate the gradient and dynamically adjust the step size may work very well in continuous domains and are rather low ranked in other search spaces with more complex interactions between the variables. In addition there may be constraints on the decision variables which complicate variation operators such as crossover or mutation.

**Fitness Function**

The fitness function is a formalization of the objective that transforms elements of the solution space $S$ into the real-valued domain of degree $n$.

$$f(x) \rightarrow \mathbb{R}^n \text{ with } x \in S$$

A decision has to be made on the value of $n$. The applicability of available methods changes quite drastically for $n = 1$, $n = [2, 3]$, and $n > 3$ which relate to the fields of single-objective, multi-objective, and many-objective optimization [ITN08]. The option to reduce a problem with multiple objectives to a single objective is possible in principle by computing the dot product of the fitness vector and a weights vector. This scalarization approach however is limited to identify only solutions as optimal that lie on the convex hull of the optimal Pareto front. Additionally, a decision on the weights for the individual objectives has to be made.

The impact of this step are rather large as the methods are entirely different and the expected results are different. In addition, approaches that consider multiple objectives usually return a set of solutions that are *non-dominated* to each other. Thus, a final selection of a single solution always has to be performed afterwards in case this is required.

**Algorithm Selection**

Especially, in research projects when the problem formulation is new and little is known about the variety and difficulty of new problem instances a DSS may be very helpful. In the framework of a DSS multiple algorithms are benchmarked against each other on a number of test cases and first results can be compared. Potentially, there is a dominating solution approach already that can be further improved through customization and specialization to the target problem. Otherwise, first insights might be obtained to which degree different algorithms and their search concepts affect the outcome quality. Potentially,

hints may be obtained that finding solutions to some group of instances is improved by having some sort of crossover, while some other group of instances is solved very well through simple hill climbing [BWA18].

For a formalized problem definition consisting of a solution representation and a fitness function several approximate algorithms exist. Often these algorithms possess a number of parameters, which create a large space of possible algorithm instances. These vary in their ability to achieve good solutions in short time. Typically, for each instance of the problem, a different algorithm instance may be optimal.

In algorithm selection, the "best" instance has to be selected for the concrete problem instance. The definition of what may be best is not as simple and basically expands into the domain of short runtime and good quality. If we fix one of these domains then it becomes quite clear, the algorithm instance that delivers the best quality in the given runtime or the algorithm instance that uses least time to achieve the desired quality. A further complication is that approximate algorithms are often stochastic thus runtime and quality both have to be seen as random variables distributed to an a priori unknown distribution.

**Solution Selection**

Algorithm instances may evaluate a number of different solutions often presenting a final best solution or a set of best solutions, e.g. those of similar quality or non-dominated solutions in multi-objective search. Sometimes minor criteria which were not added to the problem definition may also lead to a selection decision in that one solution is more preferable over another. This can happen when it would be difficult to formalize or when a bias towards such criteria should be avoided in the search.

## 5.3 Use Cases

The assumption is that the experts approach a decision support system with a new problem instance that has not be solved before [BWA16]. We aim to focus on three use cases that are relevant for experts in heuristic optimization.

1. Understanding the new instance
2. Solving the new instance
3. Document results and learn from past experiments

There is some overlap in these use cases as understanding the instance and solving it are closely related. For instance, when analyzing results of specific solvers, e.g. population-based algorithms or iterated local search variants users may be able to gain a better understanding of the instance also. The successful use of a population suggests that exploration is more vital concept and that there is some global super structure which can be exploited through combination in the population. On the other hand, if single-solution metaheuristics perform well, it may suggest just the opposite, that there is hardly a super structure such as a "big-valley" [OV16]. Combining fitness landscape analysis with performance data enables to complete the picture of similarity between problem instances.

**Understanding**

Identifying similar problem instances is one of the key interests of optimization experts. One of the main strengths of adaptation based on previous experience is to find a similar or related case and decide based on the obtained information there. Thus, it is the aim to identify if a new instance is part of a cluster and to which other instances it is most closely related. In a decision support system visualization of the space of problem instances are highly interesting. Typically, these spaces consist of the high-dimensional characteristic vector that we obtain through landscape analysis (cf. Section 3) and thus some mapping methods from high-dimensional to two dimensions have to be used, e.g. principal components analysis (PCA), multi-dimensional scaling (MDS), self-organizing maps (SOM), and t-distributed stochastic neighbor embedding (t-SNE) are commonly referred to.

An important part in such an embedding of high-dimensional spaces is the actual characteristics that should be used. Naturally, characteristics should be normalized before being embedded in order to avoid a characteristic with a

bigger range become more important. But even if such normalization is being applied, a range of features that measure basically the same characteristic may also bias the result. Thus, correlations among such features need to be studied, as was done in Section 4, in order to identify a set of characteristics that are, at least not linearly, related.

Furthermore, solutions of the problem instance provide another means of understanding. Networks of solutions may be formed, for instance by creating a graph in which an edge denotes a certain probability to move from one solution to another. These, so called, local optima networks (LON) are created by considering locally optimal solution, their quality and distance in the solution space. Nevertheless, these networks may become huge, still in the analysis of certain centrality characteristics some insights are gained [OVDT14]. Such LON have been visualized with the help of multiple dimensions. The size of a node is directly proportional to the size of its associated basin of attraction, while the colore of a node is related to its quality or fitness. Visually inspecting such a network might quickly reveal some important characteristics, for instance whether the high quality solutions are also those with the bigger basins of attraction or whether the opposite is the case. Creating LON does have a computational disadvantage as they are very expensive to compute and a thorough exploration of the solution space has to be performed. It could be said, that after a LON has been built, solving the instance is not of interest anymore as most of the interesting solutions have already been found.

**Solving**

Experts seek to identify good solutions to problem instances, either by randomly initializing a starting solution or by some predefined starting point. Especially, the later case is still mostly unexplored, and interactive optimization systems are not as well researched. However, it presents a chance for a modern and interactive approach to solving. A simple "primer" heuristic could quickly reveal some good solutions that are further improved by more complex heuristics. Also, in visualizing the network of solutions experts could choose to set starting solutions such as to intensify the search in a certain region. For this, two different seeding strategies are identified:

- Cloning
- Sampling

Solutions that are *cloned* are directly used within the starting configura-

tion of the algorithm. A single solution algorithm may choose from the set of seeding solutions, while a population-based algorithm may fill its initial population with members of that set. Unless, the set of seeding solutions is very large, it is a strategy that is more viable for single-solution metaheuristics that aim to intensify the search around a certain solution. When *sampling* from the set of seeding solutions only some components of those solutions are used. The starting configuration of the algorithm consists of a certain mix of those components. This strategy may be more viable for population-based metaheuristics, as it is easier to generate a diverse set of solutions for the initial population. Single-solution metaheuristics would be able to intensify the search in some sub-region of the solution space that is spanned by the possible combinations of the solutions in the seeding set.

While experts may be familiar with some algorithms and know their parameters and associated effects quite well, it would be quite demanding to require an understanding of each and every algorithm that has been described and implemented. The idea is that the creator provides some suggestions on the parameterization and "hand-selects" a set of instances that may be efficient on different instances. These would then be treated independently. It would not be wrong to consider some meaningful, but basic adaptation of those parameters, for instance depending on the problem size.

The provided algorithm instances could then need to be ranked by the decision support system so that experts may get some a priori feedback. It would be beneficial to have an estimation of the expected runtime to reach a certain target value, or vice-versa the expected target value given a certain runtime. Recommendation algorithms as have been applied in Section 4 can be used to process the data and suggest suitable solvers.

**Documenting and Learning**

Finally, a decision support system should enable uploading data such as problem instances, FLA characteristics, algorithm runs and solutions. This documents the progress and efforts that have been attempted at solving and may help improve the recommendation in the future when again a similar instance should be solved.

## 5.4 Software Systems

In the course of my career I have worked in several research projects and where I shaped the architecture and usage scenarios of several software systems. These systems may be coupled and put to use in real-world problem solving. In this section I would like to describe the systems in a bit more detail and how they fit into the research model.

### 5.4.1 Sim# - Discrete Event Simulator

Modeling real-world processes may be a rather complex task. As has been explained previously, simulation is one of the main modeling methodologies to describe such processes. To support modeling real-world systems, I started the discrete event simulator Sim#[11]. Sim# can be described as a port of the functionality described by SimPy[12] from the "Python world" to the C# programming language and the ".NET world". It is actively maintained open source software and has been successfully put to use, for instance to simulate the inner processes of a machine tool. There are two notable difference between SimPy and Sim#. First, SimPy may use only `double` as time, while in Sim# the `Timespan` class may be used. However, usage of doubles was added through a set of methods called the "D-API" as these all have a "D" appended to the name. The second case concerns when a process faults due to external reasons, for instance due to another process. SimPy may inject an exception into a generator, i.e., a process method, which is not possible in C#. Thus, Sim# processes that are expected to be interrupted have to call a specific method to indicate that the fault has been handled. The core concepts of Sim# are:

- Events
- Processes
- Resources

**Events** are objects that hold certain data and which may be added to the global event queue. Each event provides a list of callbacks which are executed when the event is processed. It has a lifetime that translates into the states *Alive*, *Triggered*, and *Processed*. It is alive when it has not yet been added to the global event queue and becomes triggered once it is stored there. When the event has happened it becomes processed and all callbacks are called.

[11] `https://github.com/abeham/SimSharp`
[12] `https://simpy.readthedocs.io/`

**Processes** are event generators. A process may be implemented as a method in the C# programming language. It yields a series of events. After a yield the process is suspended and will be continued when the event that was yielded is processed. This simple concept enables a concise description of a process. Processes may also spawn sub-processes and can be waited upon as for each process an event is created that will be triggered and processed when the process does not generate further events.

**Resources** are additional event queues that are used to synchronize processes. In the most simple case a resource can be requested or consumed by a process, the request itself is an event that can be yielded. Once the request is granted the process may continue. It may hold that request for as long as it likes, but should eventually release it. Other processes that require the same resource may have to wait for capacity to be available again. In general we may categorize resources according to the following criteria:

- **Spectrum** - Discrete or Continuous
- **Mixture** - Homogeneous or Heterogeneous
- **Contract** - Lease or Consume

The spectrum is most characteristic to distinguish between resources. Discrete resources usually consider a finite number of entities which may be represented by a discrete number or instance of an object in a "list-like" data type. On the other hand, continuous resources usually just capture the total quantity in a continuous number.

Continuous resources implicitly assume a single homogeneous entity of varying size. However, discrete resources may either be homogeneous or heterogeneous. In the case of a heterogeneous resource the entities have further, potentially unique properties, while for a homogeneous resource all entities are exactly alike and only their number is of interest. For instance, a workforce may be homogeneous if the individual workers are assumed to be replaceable with one another for the tasks considered in the model. However, if the workers' different skills and attributes are taken into account a resource that represents heterogeneous entities has to be used. Likewise, in a logistics scenario, transport vehicles may be considered homogeneous, e.g., all having the same capacity, or heterogeneous, e.g. with different capacities among the vehicles. Usually, modelling scenarios with heterogeneous resources adds the additional complexity that the selection of the exact entity has to be defined and which may be more complex than random or first-in-first-out (FIFO).

Finally, the contract category describes whether a certain amount or quantity of the resource is leased and has to be returned or whether it is consumed. For instance, a worker will usually be modelled as being leased, while a warehouse may be a resource that allows stocking and consuming items. A resource that leases its entities is slightly more complex as it has to track and match the leaseholders which is not necessary when entities may be fully consumed. In Sim# there exist a few standard resources which are given together with the categorization above:

- *Resource* - Discrete, Homogeneous, Lease
- *ResourcePool* - Discrete, Heterogeneous, Lease
- *Store* - Discrete, Heterogeneous, Consume
- *Container* - Continuous, Homogeneous, Consume

A *Resource* contains a discrete number of anonymous entities that can be requested and which have to be released back to the resource eventually. It employs a FIFO queue to arrange requests. There exist variants in form of a *PriorityResource* and a *PreemptivePriorityResource* to which requests with a certain priority may be made in both cases. Preemptive resources may additionally retract a processes' lease prematurely. The process that holds the lease is interrupted and has to handle the preemption or fault otherwise.

The *ResourcePool* is similar to a *Resource*, but consists of identifiable entities. This type is not part of SimPy and has been introduced only in Sim#. This may be useful, for instance when modeling a pool of employees with their individual characteristics, e.g., qualifications. An entity from the resource pool may only be borrowed for some time and has to be returned. Requests to a ResourcePool may specify a filter to define the properties of the individual to be requested (e.g. some qualification).

A *Store* contains a discrete set of heterogeneous items that can be added to and removed from. Stores may have a maximum capacity, and a so called *FilterStore* exists to retrieve items that fulfill some criteria. Store and ResourcePool are very similar, however in a Store the item does not need to return (consume instead of lease). A *PriorityStore* exists in which items have some form of priority and they are consumed in priority order, while still each put and get operation is executed in FIFO order.

A *Container* contains a continuous amount of some substance. Again, the substance may be stocked in the container and consumed. A PriorityContainer in which requests are prioritized would be thinkable, but a use case for such a

resource has not yet emerged. In any case, it is simple to extend the standard resources given that the code of all resources is open source.

In Listing 5 two processes are shown representing a certain chef that prepare some meal. Both use a common resource (oven). The result of the model is displayed in the last lines of the listing. The request events implement the `IDisposable` pattern and are thus released automatically after the using block.

In general, processes should be created that are parametrizable such that a chef can be modeled using only one process that differs in the parameters. As can be seen both chefs in Listing 5 perform the same basic steps, but differ in the time that is required. In a production or logistics scenario the processes would represent processing at a machine or some transport activity. A clear advantage of the SimPy/Sim# modeling approach is that the processes can be described without having to interact with the simulation framework a lot. The interactions are mostly to call the request and release methods of resources and several methods of the environment object.

Coincidentally, in the example in Listing 5 both cooks could finish in less time when *ChefA* got the oven before *ChefB* even if the latter had to remain idle for one time unit (or start one time unit later). This opens the world of scheduling problems for which Sim# is a suitable evaluation framework. In Listing 6 a simple deterministic permutation flowshop is simulated. The class `PermutationFlowshop` creates a default instance of 3 machines and 20 jobs. The method `SimulateShop` expects an order of the jobs. The first job in the sequence is processed first and so forth. It would be fairly simple to extend such formulations regarding "no-wait" conditions, i.e., that the job may release the current machine only after it has acquired the next.

Listing 5: Two processes in Sim# 3.1.1 and the corresponding output

```
1   static IEnumerable<Event> ChefA(Simulation env, Resource oven) {
2   env.Log("{0,3} A starts to prepare the meal.", env.NowD);
3   yield return env.TimeoutD(3.0);
4   using (var req = oven.Request()) {
5   env.Log("{0,3} A wants the oven", env.NowD);
6   yield return req;
7   env.Log("{0,3} A got the oven", env.NowD);
8   yield return env.TimeoutD(2.0);
9   env.Log("{0,3} A finished cooking, leaving the oven", env.NowD);
10  }
11  yield return env.TimeoutD(4.0);
12  env.Log("{0,3} A finished preparing.", env.NowD);
13  }
14
15  static IEnumerable<Event> ChefB(Simulation env, Resource oven) {
16  env.Log("{0,3} B starts to prepare the meal.", env.NowD);
17  yield return env.TimeoutD(2.0);
18  using (var req = oven.Request()) {
19  env.Log("{0,3} B wants the oven", env.NowD);
20  yield return req;
21  env.Log("{0,3} B got the oven", env.NowD);
22  yield return env.TimeoutD(5.0);
23  env.Log("{0,3} B finished cooking, leaving the oven", env.NowD);
24  }
25  yield return env.TimeoutD(2.0);
26  env.Log("{0,3} B finished preparing.", env.NowD);
27  }
28
29  static void Main(string[] args) {
30  var env = new Simulation();
31  var oven = new Resource(env);
32  env.Process(ChefA(env, oven));
33  env.Process(ChefB(env, oven));
34  env.Run();
35  }
36
37  // Output:
38  0 A starts to prepare the meal.
39  0 B starts to prepare the meal.
40  2 B wants the oven
41  2 B got the oven
42  3 A wants the oven
43  7 B finished cooking, leaving the oven
44  7 A got the oven
45  9 B finished preparing.
46  9 A finished cooking, leaving the oven
47  13 A finished preparing.
```

Listing 6: Permutation Flowshop Simulation using Sim# 3.1.1

```
public class PermutationFlowshop {
  private int M, J; // number of machines, jobs
  private List<List<int>> pij; // processing times per machine, job
  public int Jobs { get { return J; } }
  public int Machines { get { return M; } }

  public PermutationFlowshop() {
    pij = new List<List<int>> {
      new List<int> { 54, 83, 15, 71, 77, 36, 53, 38, 27, 87,
                      76, 91, 14, 29, 12, 77, 32, 87, 68, 94 },
      new List<int> { 79,  3, 11, 99, 56, 70, 99, 60,  5, 56,
                       3, 61, 73, 75, 47, 14, 21, 86,  5, 77 },
      new List<int> { 16, 89, 49, 15, 89, 45, 60, 23, 57, 64,
                       7,  1, 63, 41, 63, 47, 26, 75, 77, 40 }
    };
    M = 3;
    J = 20;
  }

  public PermutationFlowshop(List<List<int>> proc_times) {
    pij = proc_times;
    M = pij.Count;
    J = pij[0].Count;
  }

  public double SimulateShop(IEnumerable<int> jobs) {
    var env = new Simulation(); // create a new Sim# simulation
    // create the M machines
    var machines = Enumerable.Range(0, M).Select(x => new Resource(
        env)).ToArray();
    // add the jobs in order given by the permutation
    foreach (var j in jobs)
      env.Process(Job(env, j, machines));
    env.Run(); // run the simulation
    return env.NowD; // return the simulation time (last finish time)
  }

  private IEnumerable<Event> Job(Simulation env, int j, Resource[]
      res) {
    // for each of the M machines in order from 0 to M-1
    for (var m = 0; m < M; m++) {
      // create a new request for the next machine
      using (var req = res[m].Request()) {
        yield return req; // wait upon obtaining the machine
        // process the job j on machine m
        yield return env.TimeoutD(pij[m][j]);
      }
    }
  }
}
```

### 5.4.2 HeuristicLab Programmable Problem

The HeuristicLab optimization environment features an elaborate graphical user interface (GUI) that allows students, researchers, and practitioners explore heuristic optimization. It contains many pre-defined problems such as vehicle routing variants and pre-defined algorithms such as variants of genetic algorithms to name just a few.

One of the main contributions that I initiated and implemented together with my colleague Michael Kommenda is the programmable problem. This enables to define a new optimization problem in the GUI similar to how e.g. the CPLEX Optimization Studio or the MiniZinc IDE allows defining mathematical programming models. The basic interface of the single-objective programmable problem in HeuristicLab is given in Listing 7.

Listing 7: Interface of the programmable problem in HeuristicLab 3.3.15

```
1 bool Maximization { get; }
2
3 void Initialize();
4
5 double Evaluate(Individual individual, IRandom random);
6
7 void Analyze(Individual[] individuals, double[] qualities,
      ResultCollection results, IRandom random);
8
9 IEnumerable<Individual> GetNeighbors(Individual individual, IRandom
      random);
```

The user has to define an `Initialize` method in which the solution representation and boundary constraints on the decision variables are defined. This method defines the problem instance is and is called upon successful compilation. In the `Evaluate` method the user retrieves the solution configuration in form of an `Individual` as well as a random number generator instance in case a stochastic optimization problem is to be modeled. This method has to return a double value that depicts its quality. The `Maximization` property defines whether this quality is to be minimized or maximized. The `Analyze` method is similar to a callback after each iteration in which the user may calculate custom results, e.g. of the best-found solution so far. Finally, trajectory-based methods require the definition of a neighborhood. This is not always simple to achieve in a generic fashion. The `GetNeighbors` method allows the user to specify the neighborhood by returning all solution configurations that are neighbors to a certain solution. This approach is generic and works with

all algorithms that accept some form of neighborhood definition, but has the downside that no partial evaluation is possible. Thus, each neighbor has to be evaluated as if it was a full solution. Again a random number generator is given to this method in order to allow stochastic sampling of the neighborhood.

While Sim# is a framework for creating deterministic or stochastic simulation models to represent, the programmable problem is an interface to solve those models using algorithms. The model itself is seen as a black box. For instance, in the `Evaluate` method in Listing 7 such a model could be parameterized and started. The result of that model could then be returned. Listing 8 shows an implementation of the permutation flow shop problem as a programmable problem formulation reusing the Sim# model from Listing 6. Instead of computing the start times of the operations manually, the simulation engine will perform the necessary steps.

Listing 8: Permutation Flowshop Programmable Problem in HeuristicLab

```
using System;
using System.Linq;
using System.Collections.Generic;
using HeuristicLab.Core;
using HeuristicLab.Data;
using HeuristicLab.Encodings.PermutationEncoding;
using HeuristicLab.Optimization;

namespace PFSP {
  public class PFSPDefinition : CompiledProblemDefinition,
                                ISingleObjectiveProblemDefinition {
    private PermutationFlowshop flowshop;
    // objective is to minimize makespan
    public bool Maximization { get { return false; } }

    public override void Initialize() {
      // processing times are parsed from the instance
      flowshop = new PermutationFlowshop();
      // encoding is created
      Encoding = new PermutationEncoding("p", length: flowshop.Jobs,
                                         type: PermutationTypes.
                                             Absolute);
    }

    public double Evaluate(Individual ind, IRandom random) {
      return flowshop.SimulateShop(ind.Permutation("p"));
    }

    public void Analyze(Individual[] individuals, double[] qualities,
                        ResultCollection results, IRandom random) { }
    // for move-based algorithms, all possible swap2 neighbors are
        returned
    public IEnumerable<Individual> GetNeighbors(Individual ind,
        IRandom random) {
      var p = ind.Permutation("p");
      foreach (var move in ExhaustiveSwap2MoveGenerator.Apply(p)) {
        var neighbor = ind.Copy();
        var perm = neighbor.Permutation("p");
        Swap2Manipulator.Apply(perm, move.Index1, move.Index2);
        yield return neighbor;
      }
    }
  }
}
```

### 5.4.3 PPOVCockpit - Production Planning and Visualization

So far we have covered how Sim# supports researchers and practitioners in creating a scientific model and detailed how the programmable problem is used to embed such a model within the HeuristicLab optimization environment. A missing link in terms of systems is an interface to the real-world and to software systems typically found in real-life such as enterprise resource planning (ERP) or manufacturing execution systems (MES).

In 2009 I started the development of the PPOVCockpit. The PPOVCockpit is also based on HeuristicLab and was specifically created to interface with real-world systems. Its heart is a data model that abstracts many entities, which also occur in ERP systems such as production orders, workplaces, materials, etc. The basic data model is rather generic and can be extended to include special properties. For instance, the generic material has an identifier, dimensions, volume, and weight. A steel slab, which is a derivative of the generic material, then adds a temperature property among others. See Figure 39 for a snapshot of the data model used within the PPOVCockpit.

In 2018 the PPOVCockpit is installed in several companies in Upper Austria and provides functionalities such as performance measurement, quality control, and its data is used to parametrize optimization models such as steel stacking, intralogistic transport, or warehouse assignment. There have been many contributors from the HEAL research group to this software. A full list of contributors would be too long, but Johannes Karder and Sebastian Raggl deserve to be mentioned.

The PPOVCockpit supports the process of acquiring data, visualizing and transforming data and thus the parameterization of the scientific models. It also supports implementing the solutions which may be deferred to human operators that use the result of the software or provided to autonomous systems in terms of an API that can be called when it is run without GUI. The user interface is shown in Figure 40.

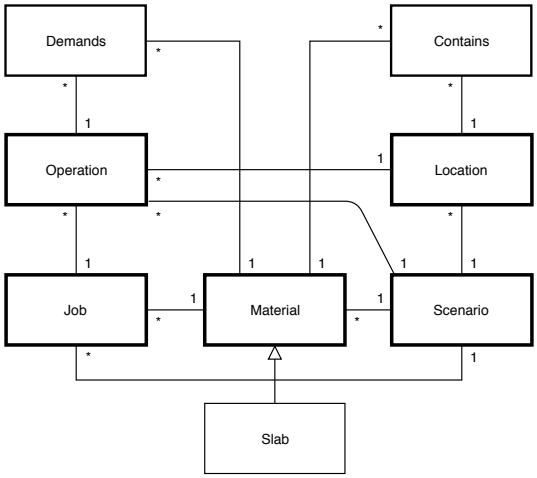



Figure 39: A snapshot of the generic data model that includes a specialization of the *Material* type in form of a *Slab*.

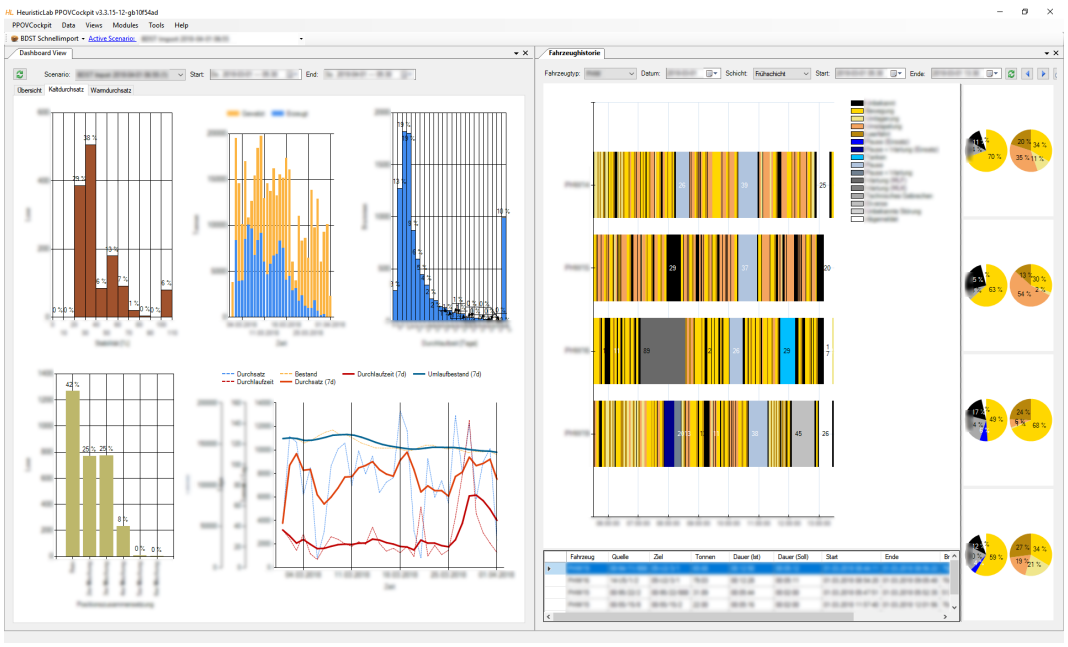


Figure 40: A screenshot of the PPOVCockpit showing performance visualizations in the steel logistics case.

# 6 Conclusions

"XXX TODO XXX"

This Author

The thesis at hand constitutes a study within the topic of fitness landscape analysis and its application in algorithm selection. I imagine this work to be important in the transition of technology created and maintained by human experts to technology that is self-maintaining and runs unobserved. Naturally, such a thesis can only cover a certain part of this process, in this case, the automated selection of suitable algorithms through the analysis of fitness landscapes. A range of open technologies are yet to be developed and methods are yet to be created which I also attempt to summarize in the remaining sections of this conclusion.

## 6.1 Summary of the Obtained Results

There are three main contributions to the state of the art that are described in this thesis in Sections 3 to 5. First, a new kind of exploratory landscape analysis method, termed "directed walks", was introduced and analyzed in detail on problem instances of the quadratic assignment problem. Three features have been derived from the resulting graphs of these walks and which are used to characterize the problem instances. It has been shown that more paths lead to more precise characterizations and about 100-200 paths have been found sufficient to identify problem instances accurately (see Table 8). It has been discussed that this new walk is suitable for integration into metaheuristic algorithms as it is part of a solution improvement procedure, although there is still research necessary. Previously described walks are not as suitable for such an integration as their goal is to cover a large part of the search space rather than identify good solutions. Directed walks may achieve both, although at a slightly higher cost. In addition to the three new features, existing features have also been analyzed. It has been found that features from *information analysis* are also suitable and can be applied as characteristics. A slight bias was found in information analysis characteristics and a remedy for that bias was proposed (see Figure 21). The effect of the bias was however insignificant and both the regular and symmetric features performed about the same for the practical task of identifying problem instances (compare IALREG and IALSYM in Table 8).

In a second contribution two larger studies have been conducted on algorithm selection applied to the quadratic assignment problem and its generalized cousin. Although the problems share common properties, they differ significantly from the perspective of a metaheuristic as the first uses a permutation while the second needs to be encoded using a more general discrete vector encoding. Algorithm instances have been implemented as described in the literature and even commercial solvers have been added to the benchmark. In both studies a combined solver using a one nearest neighbor could outperform any of the individual algorithms. For the quadratic assignment problem and a target of 5% the combined solver achieved very good performance and outperformed the best individual solver by a large margin (92% vs 58% of the instances in class 1) as shown in Table 15, but also for the harder 1% target the algorithm selection outperformed the best individual solver (53% vs 42% as given in Table 16). For the generalized quadratic assignment problem, as shown in Table 21, the combined solver was slightly better than the best single algorithm with respect to instances where it achieved rank 1 (41% vs 40%), but showed much less unsuccessful attempts (17% vs 31%). Thus, the combined solver was more robust than the single solver. Additionally, the algorithm selection problem was analyzed graphically and a new visualization in form of a scatter plot with stacked markers was devised (see Figures 32 and 37). In analyzing the performance of algorithms through correlations among each other and with landscape features it was found that certain landscape properties arising out of specific problem configurations might favor algorithms that employ a population and crossover instead of local search. Still, it must be concluded that more research is necessary to confirm or refute such a hypothesis.

In a third contribution relevant software tools were highlighted that have been created as part of the thesis. A simulation kernel was ported from Python to C# and extended. Examples are given how complex problems from the domain of scheduling could be evaluated using this simulation kernel. In addition, an interface was described that allows users to formulate optimization problems for metaheuristic algorithms and it was shown how this could be combined with the simulation kernel to pursue simulation-based optimization applications. The case of a decision support system for metaheuristic optimization was discussed and three use cases have been presented. Finally, a software system which has been used in practice to store data from real-world production systems was shown, and the data model has been introduced. Applications using this software system have been discussed.

## 6.2 Outlook and Future Work

Similar to the three main contributions above I also like to summarize and extend the future work in these topics. First, the topic of fitness landscape analysis is explored more intensively already, but the integration within metaheuristic algorithms is a new field. Nevertheless, it needs to be studied how the dynamics of a search algorithm influence the quality of the characteristics from such walks. For instance, a highly similar set of solutions between which directed walks are performed would lead to rather short walks. The characteristic data would thus consist of both long and short walks. Methods need to be devised how such data can be made use of and how the performance of landscape identification is degraded when data quality drops. In this regard, I think the potential of inverse directed walks as introduced as part of Section 3 are more interesting. As can be observed in Figure 17 on page 67, locally optimal solutions may even be improved slightly in a couple of steps of an inverse directed walk. Thus, this type of exploratory walk would contribute to the quality of solutions that an algorithm found. Also these walks are of roughly similar length, regardless of the state of the population and, as was observed in Tables 7 and 8 on page 85, are best suited to provide information for landscape identification. On the other hand, they are also the most costly to compute (Table 9) as the neighborhood is larger than in regular directed walks. However, in the presence of an efficient delta evaluation this can be worthwhile. The author's MemPR algorithm that achieved $2^{nd}$ place at the GECCO 2016 in Denver in the combinatorial black box optimization benchmark includes such a type of walk. Still, it did not yet use this information as part of an "in-situ" decision on parameter adaption or algorithm for a post-optimization phase. Further algorithms that include inverse directed walks are not known to me.

The topic of algorithm selection provides a challenging field for future research. While landscape analysis and feature-based algorithm selection become more pervasive in the scientific literature, the broader topics of optimizing the algorithm portfolios and problem instance libraries themselves are still not as widely considered. These topics need more attention. In my thesis I have built strongly on personal experience in devising the various algorithms that are used to form a portfolio. However, for a future application an automation of that process would be of value. Systems are needed that automatically benchmark algorithm instances against each other and decide on the keepers that make up

a successful portfolio. But this is not the only important aspect. The success of such a portfolio depends on three ingredients: (1) the algorithm instances, (2) the problem instances, and (3) the recommender or selector, respectively schedule builder. Especially, the problem instances are a crucial part which greatly influences the computational effort to manage such a portfolio. In addition, when the range of problem instances that are available to the portfolio is only limited to a certain "kind", algorithm instances may not become part of the portfolio that would outperform the others on another "kind". Finally, the recommender is important in such a way that when the performance of algorithm instances may not be discriminated well enough, only an average performance might be reached. In addition, missing data poses a real-world problem that has been ignored so far. But it is important to identify algorithm instances to recommend when we do not know all their performance on all the instances. It is unknown how the recommendation algorithms drop in performance when the data contains missing values.

The topic of algorithm selection still poses a lot of open questions. I have pursued a k-nearest neighbor approach with a fixed setting of $k = 1$ and formulated the problem as a classification task (Tables 15 and 21). However, a regression-based approach would also be possible, and there are many more machine learning approaches that can be applied and compared. In addition, visualization is a very important aspect. The newly developed graphs as shown in Figure 32 and 37 may show a lot of information. However, it is still an open task to also describe the clusters and sub-clusters of the problem instances such that those are not merely anonymous points. To devise new interactive visualizations would be an interesting branch for future development.

Finally, an interesting topic for future work would be the standardization of benchmarking efforts and the increased compatibility between results generated with different frameworks and the increased availability of open source implementations. It is still cumbersome to this date to find open source implementations of algorithms and to perform a thorough comparison with state of the art performance. The description of algorithms in a prosaic form often falls short of the specificity of a real implementation in a real programming language. Reimplementations always bear the possibility of introducing additional errors and may invalidate conclusions and research efforts. In summary, I hope to have presented some concrete opportunities for future work and further research.

## 6.3 Postscript

Digitalization and subsequently automation are major keywords of the late 2010s, which fill the topic lists of major national and EU funded research programs. It is expected, given the progress that digital companies such as Google or Facebook have achieved, that this process results in new abilities to analyze, understand, and predict performances and relationships in real-world systems such as manufacturing, healthcare, or agriculture. The vision is that new digital abilities and systems enable faster and better decision making and control. The hope is that this leads to improvements in efficiencies, as well as new products and services which may bring together producers, logistics providers, and consumers closer than ever.

Still far from a widespread realization of this vision, the current focus is to digitalize the real-world processes that naturally are reluctant to change. I said half-jokingly once, that the presence of telephones is still one of the major obstacles to Industry 4.0. But therein lies the truth, that, in a digital world humans are expected to collaborate using digital tools. Telephones and paper notebooks that are ever so present in these days are highly analogue devices. The research agenda to create algorithms and systems that can be called using a phone or which can be given hand-written notes as forms of interaction is rather small. Even though some progress, most notably by Google, has been made in that direction already. The typical man-machine interaction still occurs with the user interfaces that we have all become acquainted with.

A further challenge remains in synchronizing the digital and the analogue world. Unless all physical items become internet of things (IoT) devices that area physically co-located, the digital and physical world have to be kept synchronized. Research efforts are directed towards the creation of, so called, digital twins that synchronize their properties and states with their physical sibling. In a broad sense, many such twins are already in place and it is unthinkable how business would operate without them. Enterprise information systems such as ERP, MES and warehouse information systems provide a digital copy of the real system. However, it is expected that even more state and information is maintained digitally and partly in centralized databases, partly in decentralized storages and in close proximity to the physical item.

The work in this thesis becomes more and more relevant the more progress is made in the digitalization process. In the emerging situations of the digitalized world, automation becomes a greater focus. In present day operations,

automation is still mostly concerned with physical processes and their sensors that are queried by programmable logic controllers (PLCs) and provide quick decisions for machine actuators. In the future, it is expected that algorithms are even more important in controlling digital processes. In the financial sector, this has been achieved already to a larger extent in that many transactions and even whole funds are managed by algorithms. This also has raised new "issues" in that the length of network cables at stock exchanges may be a discriminating factor as those algorithms connected to the stock exchange systems with longer cables are slower to answer to a potential trade.

In manufacturing, for instance, scheduling algorithms control the material flow, not by directing material handling equipment, but by managing the precedences of jobs that are to be executed. Industry 4.0 is still tied on the point whether these algorithms are to be successfully employed in a central manner as has been attempted in the previous decade or whether decentralized approaches need to emerge; research is still ongoing.

In my personal experience in automating the crane operations at a continuous caster in the steel industry, devising new digital processes that replace their analogue predecessors is immensely more complex and often requires formulating rules much more precisely than before and creating new data that has to be maintained. In addition, the vast decision spaces with constraints that are sometimes hard to formalize and get input data of, the presence of many conflicting objectives where preference varies according to the "current situation" still challenge our abilities of controlling real-world systems with digital models. Additional research efforts in the optimization of dynamic and uncertain systems are still necessary. In this light, algorithm selection seems to be a minuscule challenge, but mostly, because such topics are still ahead of their time. To me it is clear that the progress in digitalization and the ability to improve the scientific modeling and solving of real-world decision situations, the topic of algorithm selection becomes ever so important. Above all, the time to achieve decisions is crucial, and again judging from personal experience, there is a huge difference between a decision in 1 second vs in 10 seconds, as the world changes. Time and quality, which are both important to algorithm selection, are the key elements of successful digital decision makers.

# References

[ABCC07] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007. → page 12

[AD04] Enrique Alba and Bernabé Dorronsoro. Solving the Vehicle Routing Problem by Using Cellular Genetic Algorithms. In Jens Gottlieb and Günther R Raidl, editors, *EvoCOP 2004: Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science (LNCS 3004)*, pages 11–20. Springer, Berlin, Heidelberg, 2004. → page 1

[AD08] Enrique Alba and Bernab Dorronsoro. *Cellular Genetic Algorithms*. Springer US, 1st edition, 2008. → page 29

[AH05] A Auger and N Hansen. Performance evaluation of an advanced local search evolutionary algorithm. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC)*, volume 2, pages 1777–1784, sep 2005. → page 111

[Alb05] Enrique Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley, 2005. → page 27

[ALN13] Enrique Alba, Gabriel Luque, and Sergio Nesmachnow. Parallel metaheuristics: Recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48, 2013. → page 27

[Alt94] Lee Altenberg. The Evolution of Evolvability in Genetic Programming. In Jr. Kinnear and E. Kenneth, editors, *Advances in Genetic Programming*, pages 47–74. MIT Press, 1994. → page 28

[ARR07] Renata M Aiex, Mauricio G C Resende, and Celso C Ribeiro. TTT plots: a perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366, 2007. → page 51

[AST09] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5732 LNCS, pages 142–157, 2009. → page 53

[AT01] Enrique Alba and José M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems*, 17(4):451–465, 2001. → page 27

[AW03] Michael Affenzeller and Stefan Wagner. A self-adaptive model for selective pressure handling within the theory of genetic algorithms. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 2809, pages 384–393. Springer, 2003. → page 27

[AWWB09] Michael Affenzeller, Stephan Winkler, Stefan Wagner, and Andreas Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. New York: Chapman and Hall/CRC, 2009. → pages 26, 27, 132, and 133

[AZ98] Eric Angel and Vassilis Zissimopoulos. Autocorrelation coefficient for the graph bipartitioning problem. *Theoretical Computer Science*, 191(1-2):229–243, 1998. → page 126

[BAP15] Andreas Beham, Michael Affenzeller, and Erik Pitzer. Metaheuristic Algorithms for the Quadratic Assignment Problem: Performance and Comparison. In *Innovative Technologies in Management and Science*, pages 171–190. Springer, 2015. → page 95

[BAW17] Andreas Beham, Michael Affenzeller, and Stefan Wagner. Instance-based Algorithm Selection on Quadratic Assignment Problem Landscapes. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 1471–1478, New York, NY, USA, 2017. ACM. → pages 4, 55, 57, 71, 91, 94, 95, 104, 106, 114, 118, 119, and 153

[BB06] Thomas Bartz-Beielstein. *Experimental Research in Evolutionary Computation - The New Experimentalism*. Natural Computing Series. Springer, 2006. → pages 6, 9, and 47

[BB17] Edmund K Burke and Yuri Bykov. The late acceptance Hill-Climbing heuristic. *European Journal of Operational Research*, 258(1):70–78, 2017. → page 129

[BBLP05] T Bartz-Beielstein, C W G Lasarczyk, and M Preuss. Sequential parameter optimization. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 773–780 Vol.1, sep 2005. → page 4

[BBLP10] Thomas Bartz-Beielstein, Christian Lasarczyk, and Mike Preuss. The sequential parameter optimization toolbox. In *Experimental Methods for the Analysis of Optimization Algorithms*, pages 337–362. Springer, 2010. → page 53

[BDBV04] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–449, 2004. → page 2

[BE12] Jürgen Branke and Jawad Asem Elomari. Meta-optimization for Parameter Tuning with a Flexible Computing Budget. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, pages 1245–1252, New York, NY, USA, 2012. ACM. → page 3

[Bea94] James C Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, 6(2):154–160, 1994. → page 29

[BEG+11] Thierry Benoist, Bertrand Estellon, Frédéric Gardi, Romain Megel, and Karim Nouioua. Localsolver 1.x: a black-box local-search solver for 0-1 programming. *4OR: A Quarterly Journal of Operations Research*, 9(3):299–316, 2011. → page 139

[Ben02] Saifallah Benjaafar. Modeling and Analysis of Congestion in the Design of Facility Layouts. *Management Science*, 48(5):679—-704, 2002. → page 16

[BF02] J. Will M. Bertrand and Jan C. Fransoo. Operations management research methodologies using quantitative modeling. *International Journal of Operations and Production Management*, 22(2):241–264, 2002. → pages 157 and 158

[BGH+13] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013. → pages 4 and 53

[BGN+19] Wojciech Bożejko, Andrzej Gnatowski, Teodor Niżyński, Michael Affenzeller, and Andreas Beham. Local Optima Networks in Solving Algorithm Selection Problem for TSP. In *Advances in Intelligent Systems and Computing*, volume 761, pages 83–93. 2019. → page 54

[BKK+16] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016. → page 54

[BKR97] Rainer E Burkard, Stefan E Karisch, and Franz Rendl. {QAPLIB} – {A} Quadratic Assignment Problem Library. *Journal of Global Optimization*, 10(4):391–403, 1997. → pages 14, 82, 104, and 123

[BL17] Mosab Bazargani and Fernando G Lobo. Parameter-less Late Acceptance Hill-climbing. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 219–226, New York, NY, USA, 2017. ACM. → pages 129, 130, and 131

[Bla57] William M. Blair. President draws planning moral: Recalls Army Days to Show Value of Preparedness in Time of Crisis. New York Times, November 15th, 1957. → page 157

[BMTP12] Bernd Bischl, Olaf Mersmann, Heike Trautmann, and Mike Preuß. Algorithm Selection Based on Exploratory Landscape Analysis and Cost-Sensitive Learning Categories and Subject Descriptors. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, pages 313–320, New York, NY, USA, 2012. ACM. → page 44

[BPSV01] Mauro Birattari, Luis Paquete, Thomas Stützle, and Klaus Varrentrapp. Classification of Metaheuristics and Design of Experiments for the Analysis of Components. Tech. Rep. AIDA-01-05, Intellektik, Darmstadt University of Technology, 2001. → pages 20, 21, and 22

[BPWA17] Andreas Beham, Erik Pitzer, Stefan Wagner, and Michael Affenzeller. Integrating Exploratory Landscape Analysis into Metaheuristic Algorithms. In *International Conference on Computer Aided Systems Theory*, pages 473–480. Springer, 2017. → pages 55, 57, 71, 80, 82, 83, and 91

[BR03] Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, sep 2003. → pages 18, 20, and 21

[BS02] H. G. Beyer and H. P. Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002. → pages 1, 2, and 28

[BWA16] A. Beham, S. Wagner, and M. Affenzeller. Optimization knowledge center: A decision support system for heuristic optimization. In *GECCO 2016 Companion - Proceedings of the 2016 Genetic and Evolutionary Computation Conference*, 2016. → page 163

[BWA18] Andreas Beham, Stefan Wagner, and Michael Affenzeller. Algorithm Selection on Generalized Quadratic Assignment Problem Landscapes. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 253–260, New York, NY, USA, 2018. ACM. → pages 44, 122, 125, 127, 133, 145, 147, 148, 155, 156, and 162

[BWWA08] Andreas Beham, Stephan M. Winkler, Stefan Wagner, and Michael Affenzeller. A Genetic Programming Approach to Solve Scheduling Problems with Parallel Simulation. In *Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium*, page 5, 2008. → pages 4 and 160

[CCL13] Leandro C. Coelho, Jean-François Cordeau, and Gilbert Laporte. Thirty Years of Inventory Routing. *Transportation Science*, 48(1):1–19, 2013. → page 1

[CGLM06] Jean-François Cordeau, Manlio Gaudioso, Gilbert Laporte, and Luigi Moccia. A memetic heuristic for the generalized quadratic assignment problem. *INFORMS Journal on Computing*, 18(4):433–443, 2006. → pages 16, 123, and 124

[CLA12] Francisco Chicano, Gabriel Luque, and Enrique Alba. Autocorrelation measures for the quadratic assignment problem. *Applied Mathematics Letters*, 25(4):698–705, apr 2012. → pages 34, 35, and 42

[CM03] S Conway-Morris. The Cambrian "explosion" of metazoans and molecular biology: would Darwin be satisfied?". *The International journal of developmental biology*, 47(7–8):505–515, 2003. → page 5

[CVDS19] Christian Leonardo Camacho-Villalón, Marco Dorigo, and Thomas Stützle. The intelligent water drops algorithm: why it cannot be considered a novel algorithm. *Swarm Intelligence*, May 2019. → page 5

[CWA11] Francisco Chicano, L Darrell Whitley, and Enrique Alba. A methodology to find the elementary landscape decomposition of combinatorial optimization problems. *Evolutionary computation*, 19(4):597–637, jan 2011. → page 42

[DCDS17] Nguyen Dang, Leslie Pérez Cáceres, Patrick De Causmaecker, and Thomas Stützle. Configuring Irace Using Surrogate Configuration Benchmarks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 243–250, New York, NY, USA, 2017. ACM. → page 4

[DD99] Marco Dorigo and Gianni Di Caro. The Ant Colony Optimization Meta-Heuristic. In D Crone, M Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999. → pages 5 and 29

[DHTT05] Zvi Drezner, PM Peter M Hahn, ÉD Taillard, and Erik D Taillard. Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. *Annals of Operations Research*, 139(1976):65–94, 2005. → pages 14, 16, 82, and 104

[DLLIS11] Jérémie Dubois-Lacoste, Manuel López-Ibáñez, and Thomas Stützle. Improving the anytime behavior of two-phase local search. *Annals of mathematics and artificial intelligence*, 61(2):125–154, 2011. → page 17

[DMW99] Jonathan P K Doye, Mark A Miller, and David J Wales. The double-funnel energy landscape of the 38-atom Lennard-Jones cluster. *The Journal of Chemical Physics*, 110(14):6896–6906, 1999. → page 38

[DPA+02] Kalyanmoy Deb, Amrit Pratab, Sameer Agarwal, T. Meyarivan, A. Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. → pages 5 and 25

[dR06] Sérgio A de Carvalho Jr. and Sven Rahmann. Microarray Layout as Quadratic Assignment Problem. In *Proceedings of the German Conference on Bioinformatics (GCB), volume P-83 of Lecture Notes in Informatics*, 2006. → pages 14, 16, 82, and 104

[Dre08] Zvi Drezner. Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem.

*Computers & Operations Research*, 35(3):717–736, mar 2008. → page 16

[DVOT14] Fabio Daolio, Sébastien Verel, Gabriela Ochoa, and Marco Tomassini. Local optima networks of the permutation flow-shop problem. In *Artificial Evolution. EA 2013. Lecture Notes in Computer Science, vol 8752*, volume 8752, pages 41–52. Springer, Cham, 2014. → page 46

[Els77] Alwalid N Elshafei. Hospital Layout as a Quadratic Assignment Problem. *Operational Research Quarterly*, 28(1):167–179, 1977. → page 16

[FDG+12] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. {DEAP}: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012. → page 7

[Fie18] Jonathan E Fieldsend. Computationally Efficient Local Optima Network Construction. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '18, pages 1481–1488, New York, NY, USA, 2018. ACM. → page 46

[Fog88] David B Fogel. An evolutionary approach to the traveling salesman problem. *Biological Cybernetics*, 60(2):139–144, 1988. → pages 99, 102, and 103

[FR95] Thomas A Feo and Mauricio G C Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6(2):109–133, 1995. → page 5

[FR14] Steffen Finck and Raymond Ros. COCO (COmparing Continuous Optimizers) Software: User Documentation. Online: http://coco.lri.fr/downloads/download15.02/bbobdocsoftware.pdf, 2014. → page 51

[FRS94] Thomas A Feo, Mauricio G C Resende, and Stuart H Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42(5):860–878, 1994. → page 50

[FY83] Alan M Frieze and Joseph Yadegar. On the Quadratic Assignment Problem. *Discrete applied mathematics*, 5(1):89–98, 1983. → pages 16 and 137

[Gil62] Paul C Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the society for industrial and applied mathematics*, 10(2):305–313, 1962. → page 16

[GKL01] Zong Woo Geem, Joong Hoon Kim, and Gobichettipalayam Vasudevan Loganathan. A new Heuristic Optimization Algorithm: Harmony Search. *Simulation*, 76(2):60–68, 2001. → page 5

[GLM00] Fred Glover, Manuel Laguna, and Rafael Martí. Fundamentals of Scatter Search and Path Relinking. *Control and Cybernetics*, 29(3):653–684, 2000. → page 56

[Glo86] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computer and Operations Research*, 13(5):533–549, 1986. → pages 5 and 30

[Glo99] Fred Glover. Scatter Search and Path Relinking. In D Corne, M Dorigo, F Glover, D Dasgupta, P Moscato, R Poli, and K V Price, editors, *New Ideas in Optimization*, Advanced Topics in Computer Science, pages 297–316. McGraw-Hill, 1999. → pages 5 and 28

[GP14] Brian W Goldman and William F Punch. Parameter-less Population Pyramid. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 785–792, New York, NY, USA, 2014. ACM. → page 5

[Gri81] Andreas O Griewank. Generalized descent for global optimization. *Journal of Optimization Theory and Applications*, 34(1):11–39, 1981. → page 38

[GS09] Matteo Gagliolo and Jürgen Schmidhuber. Towards distributed algorithm portfolios. In *Advances in Soft Computing*, volume 50, pages 634–643, 2009. → page 53

[GS17] Kevin Graham and Leslie Smith. Comparing hyper-heuristics with blackboard systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO '17*, GECCO '17, pages 1141–1145, New York, NY, USA, 2017. ACM. → page 53

[HAFR09] Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. Real-Parameter Black-Box Optimization Benchmarking 2009: Experimental Setup. Technical Report inria-00362649v2, INRIA, 2009. → page 51

[HAM+16] Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tusar, and Dimo Brockhoff. {COCO:} {A} Platform for Comparing Continuous Optimizers in a Black-Box Setting. *CoRR*, abs/1603.0, 2016. → pages 47 and 48

[HB10] Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, 207(1):1–14, 2010. → page 2

[HBR+19] Viktoria A. Hauder, Andreas Beham, Sebastian Raggl, Sophie N. Parragh, and Michael Affenzeller. On constraint programming for a new flexible project scheduling problem with resource constraints. Technical report, feb 2019. → page 2

[HHL+15] Megan L Head, Luke Holman, Rob Lanfear, Andrew T Kahn, and Michael D Jennions. The extent and consequences of p-hacking in science. *PLoS biology*, 13(3):e1002106, 2015. → page 49

[HHLB11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Lecture Notes in Computer Science*, volume 6683, pages 507–523, 2011. → pages 4 and 53

[HHLBS09] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009. → pages 4 and 53

[HK01] Peter M Hahn and Jakob Krarup. A hospital facility layout problem finally solved. *Journal of Intelligent Manufacturing*, 12:487–496, 2001. → page 16

[HKB+18] V.A. Hauder, J. Karder, A. Beham, S. Wagner, and M. Affenzeller. A general solution approach for the location routing problem. *Lecture Notes in Computer Science*, 10671 LNCS, 2018. → page 2

[HM01] Pierre Hansen and Nenad Mladenović. Variable Neighborhood Search: Principles and Applications. *European Journal of Operational Research*, 130(3):449–467, 2001. → page 5

[HO01] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, jan 2001. → pages 2 and 28

[Hol75] John H Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975. → pages 2 and 5

[Hol92] John Henry Holland. *Adaptation in Natural and Artificial Systems*. MIT press, 1992. → pages 26 and 27

[Hor96] Wim Hordijk. A measure of landscapes. *Evolutionary Computation*, 4(4):335–360, 1996. → page 78

[Hor06] Gregory S Hornby. ALPS: The Age-layered Population Structure for Reducing the Problem of Premature Convergence. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 815–822, New York, NY, USA, 2006. ACM. → pages 20 and 89

[HS98] H H Hoos and T Stützle. Evaluating Las Vegas Algorithms - Pitfalls and Remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 238–245, San Francisco, CA, 1998. Morgan Kaufmann. → pages 18, 50, and 111

[HS14] Mohamed Saifullah Hussin and Thomas Stützle. Tabu search vs. simulated annealing as a function of the size of quadratic assignment problem instances. *Computers and Operations Research*, 43:286–291, 2014. → page 53

[ITN08] Hisao Ishibuchi, Noritaka Tsukamoto, and Yusuke Nojima. Evolutionary many-objective optimization: A short review. In *2008 IEEE Congress on Evolutionary Computation, CEC 2008*, pages 2419–2426, 2008. → page 161

[Jai85] Patrick Jaillet. *Probabilistic traveling salesman problems*. PhD thesis, Massachusetts Institute of Technology, 1985. → page 20

[JK02] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002. → page 112

[Kal00] Deb Kalyanmoy. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):311–338, 2000. → page 123

[Kat14] Yoshiaki Katada. Estimating the Degree of Neutrality and Ruggedness of Fitness Landscapes. In *Recent Advances in the Theory and Application of Fitness Landscapes*, pages 207–231. Springer, 2014. → page 34

[Kau14] SA Kauffman. Foreword: Statable and non-prestatable fitness landscapes. *Recent Advances in the Theory and Application of Fitness Landscapes*, 2014. → page 55

[KB57] Tjalling C Koopmans and Martin Beckmann. Assignment Problems and the Location of Economic Activities. *Econometrica, Journal of the Econometric Society*, 25(1):53–76, 1957. → pages 1 and 11

[KB78] L Kaufman and Fernand Broeckx. An Algorithm for the Quadratic Assignment Problem using Bender's Decomposition. *European Journal of Operational Research*, 2(3):207–211, 1978. → pages 16 and 137

[KE95] James Kennedy and Russel C Eberhardt. Particle Swarm Optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE Press, 1995. → pages 5 and 29

[Ken38] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938. → page 113

[KGV83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983. → page 5

[KKHT15] Lars Kotthoff, Pascal Kerschke, Holger H. Hoos, and Heike Trautmann. Improving the state of the art in inexact TSP solving using per-instance algorithm selection. In *Learning and Intelligent Optimization. LION 2015. Lecture Notes in Computer Science, vol 8994*, volume 8994, pages 202–217. Springer, Cham, 2015. → page 54

[KL13] Slawomir Koziel and Leifur Leifsson. *Surrogate-Based Modeling and Optimization*. Springer-Verlag New York Inc., 2013. → page 22

[Kot16] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. In *Lecture Notes in Computer Science*, volume 10101, pages 149–190. Springer, 2016. → pages 52 and 54

[KPWT15] Pascal Kerschke, Mike Preuss, Simon Wessing, and Heike Trautmann. Detecting Funnel Structures by Means of Exploratory Landscape Analysis. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 265–272, New York, NY, USA, 2015. ACM. → pages 38 and 40

[KS11] Daniel J Kvitek and Gavin Sherlock. Reciprocal sign epistasis between frequently experimentally evolved adaptive mutations causes a rugged fitness landscape. *PLoS genetics*, 7(4):e1002056, 2011. → page 34

[Kuh55] Harold W Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. → pages 10 and 16

[Law63] Eugene L Lawler. The quadratic assignment problem. *Management science*, 9(4):586–599, 1963. → page 16

[LDV+17] Arnaud Liefooghe, Bilel Derbel, Sébastien Verel, Hernán Aguirre, and Kiyoshi Tanaka. Towards landscape-aware automatic algorithm configuration: Preliminary experiments on neutral and rugged landscapes. In Bin Hu and Manuel López-Ibáñez, editors, *Lecture Notes in Computer Science 10197*, volume 10197 LNCS, pages 215–232, Amsterdam, Netherlands, 2017. Springer. → page 53

[LIDLP+16] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. → page 53

[LIDLSB11] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. The irace package, Iterated Race for Automatic Algorithm Configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011. → page 4

[LIS14] Manuel López-Ibáñez and Thomas Stützle. Automatic (offline) configuration of algorithms. In *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion - GECCO Comp '14*, GECCO Comp '14, pages 921–946, New York, NY, USA, 2014. ACM. → page 53

[LMS10] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated Local Search: Framework and Applications. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics: Second Edition*, volume 146 of *International Series in Operations Research and Management Science*, pages 363–398. Springer Science+Business Media, LLC, 2010. → page 30

[LO16] Per Kristian Lehre and Pietro S Oliveto. Runtime Analysis of Population-based Evolutionary Algorithms. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 435–462. ACM, 2016. → page 18

[LW06] Monte Lunacek and Darrell Whitley. The dispersion metric and the CMA evolution strategy. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 477–484. ACM, 2006. → page 39

[MBPS74] Ian I. Mitroff, Frederick Betz, Louis R. Pondy, and Francisco Sagasti. On Managing Science in the Systems Age: Two Schemas for the Study of Science as a Whole Systems Phenomenon. *Interfaces*, 4(3):46–58, 1974. → pages 157 and 158

[MBT+11] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO '11*, 2011. → pages 42 and 43

[McC76] Thomas J McCabe. A Complexity Measure. *IEEE Transactions on software Engineering*, SE-2(4):308–320, 1976. → page 49

[ME14a] Katherine M. Malan and Andries P. Engelbrecht. A Progressive Random Walk Algorithm for Sampling Continuous Fitness Landscapes. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014*, pages 2507–2514, 2014. → page 44

[ME14b] Katherine M Malan and Andries P Engelbrecht. Fitness landscape analysis for metaheuristic performance prediction. In *Recent advances in the theory and application of fitness landscapes*, pages 103–132. Springer, 2014. → pages 38 and 39

[MF97] Peter Merz and Bernd Freisleben. A Genetic Local Search Approach to the Quadratic Assignment Problem. In Thomas Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms, East Lansing, MI, USA, July 19-23, 1997*, pages 465–472. Morgan Kaufmann, 1997. → pages 101 and 103

[MF00] Peter Merz and Bernd Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4):337–352, 2000. → pages 16, 33, and 101

[MGA17] I Moser, M Gheorghita, and A Aleti. Identifying Features of Fitness Landscapes and Relating Them to Problem Difficulty. *Evolutionary Computation*, 25(3):407–437, 2017. → page 34

[MH97] N Mladenović and P Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997. → pages 29 and 99

[MH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. → page 147

[Mic99] Z Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3rd edition, 1999. → pages 2 and 26

[MMC11] Efrén Mezura-Montes and Carlos A. Coello Coello. Constraint-handling in nature-inspired numerical optimization: Past, present and future. *Swarm and Evolutionary Computation*, 1(4):173–194, dec 2011. → page 123

[MRS11] Geraldo R Mateus, Mauricio G C Resende, and Ricardo M A Silva. GRASP with path-relinking for the generalized quadratic assignment problem. *Journal of Heuristics*, 17(5):527–565, 2011. → pages 16, 129, 134, 136, and 144

[NCA11] Sergio Nesmachnow, Héctor Cancela, and Enrique Alba. Heterogeneous computing scheduling with evolutionary algorithms. *Soft Computing*, 15(4):685–701, 2011. → page 2

[OV16] Gabriela Ochoa and Nadarajen Veerapen. Deconstructing the big valley search space hypothesis. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9595, pages 58–73, 2016. → page 163

[OV17] Gabriela Ochoa and Nadarajen Veerapen. Mapping the global structure of TSP fitness landscapes. *Journal of Heuristics*, pages 1–30, 2017. → page 38

[OVDT14] Gabriela Ochoa, Sébastien Verel, Fabio Daolio, and Marco Tomassini. Local Optima Networks: A New Model of Combinatorial Fitness Landscapes. In *Recent Advances in the Theory and Application of Fitness Landscapes*, pages 233–262. Springer, 2014. → pages 33, 45, and 164

[PA12] Erik Pitzer and Michael Affenzeller. A Comprehensive Survey on Fitness Landscape Analysis. In János Fodor, Ryszard Klempous, and Carmen Paz Suárez Araujo, editors, *Recent Advances in Intelligent Engineering Systems*, chapter Chapter 1:, pages 161–191. Springer, Berlin, Heidelberg, 2012. → pages 33, 42, 43, 44, and 126

[PAB10] E. Pitzer, M. Affenzeller, and A. Beham. A closer look down the basins of attraction. In *2010 UK Workshop on Computational Intelligence, UKCI 2010*, 2010. → page 33

[PHGZ10] Artur Alves Pessoa, Peter M Hahn, Monique Guignard, and Yi-Rong Zhu. Algorithms for the generalized quadratic assignment problem combining Lagrangean decomposition and the Reformulation-Linearization Technique. *European Journal of Operational Research*, 206(1):54–63, 2010. → pages 1 and 13

[Pin16] Michael L Pinedo. *Scheduling*. Springer, Cham, Cham, 2016. → page 1

[Pit13] Erik Pitzer. *Applied Fitness Landscape Analysis*. PhD thesis, Johannes Kepler University Linz, 2013. → pages 38 and 80

[PP14] Caroline Prodhon and Christian Prins. A survey of recent research on location-routing problems, 2014. → page 1

[PR07] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007. → page 18

[PR10] David Pisinger and Stefan Ropke. Large Neighborhood Search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 399–419. Springer US, 2010. → page 5

[PT09] Luca Pulina and Armando Tacchella. A self-adaptive multi-engine solver for quantified Boolean formulas. *Constraints*, 14(1):80–116, 2009. → page 53

[Rec73] Ingo Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973. → pages 2, 5, and 28

[Rei91] Gerhard Reinelt. {TSPLIB} - {A} Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3:376–384, 1991. → page 14

[Ric76] J.R. John R Rice. The Algorithm Selection Problem. *Advances in Computers*, 15:65–118, 1976. → pages 3, 6, 47, 52, and 94

[RS75] G Terry Ross and Richard M Soland. A branch and bound algorithm for the generalized assignment problem. *Mathematical programming*, 8(1):91–103, 1975. → pages 1 and 12

[RS01] Christian M. Reidys and Peter F. Stadler. Neutrality in fitness landscapes. *Applied Mathematics and Computation*, 117(2-3):321–350, jan 2001. → page 44

[SDH11] M. Schilde, K. F. Doerner, and R. F. Hartl. Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. *Computers and Operations Research*, 38(12):1719–1730, 2011. → page 1

[SF12] Gail M Sullivan and Richard Feinn. Using effect size—or why the P value is not enough. *Journal of graduate medical education*, 4(3):279–282, 2012. → page 49

[SHP15] Kevin Sim, Emma Hart, and Ben Paechter. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary Computation*, 23(1):37–67, 2015. → pages 53 and 54

[SM73] Francisco R. Sagasti and Ian I. Mitroff. Operations research from the viewpoint of general systems theory. *Omega*, 1(6):695–709, 1973. → page 157

[SM08] K. A. Smith-Miles. Towards Insightful Algorithm Selection For Optimisation Using Meta-Learning Concepts. In *IEEE International Joint Conference on Neural Networks*, pages 4118–4124, 2008. → pages 52 and 158

[SMJGT09] Kate A. Smith-Miles, Ross J.W. James, John W. Giffin, and Yiqing Tu. A knowledge discovery approach to understanding relationships between scheduling problem structure and heuristic performance. In Thomas Stützle, editor, *Learning and Intelligent Optimization. LION 2009. Lecture Notes in Computer Science, vol 5851*, pages 89–103. Springer, 2009. → page 52

[SMvH11] Kate Smith-Miles and Jano van Hemert. Discovering the suitability of optimisation algorithms by learning from evolved instances.

*Annals of Mathematics and Artificial Intelligence*, 61(2):87–104, feb 2011. → pages 48, 52, 53, 54, and 158

[Sör15] Kenneth Sörensen. Metaheuristics–the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18, 2015. → page 5

[Sri95] R. Sridharan. The capacitated plant location problem, 1995. → page 1

[Sta02] P.F. Stadler. Fitness landscapes. In *Biological Evolution and Statistical Physics*, pages 183–204. Springer Berlin Heidelberg, 2002. → page 34

[Ste61] Leon Steinberg. The Backboard Wiring Problem: A Placement Algorithm. *Siam Review*, 3(1):37–50, 1961. → page 16

[Stü06] Thomas Stützle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3):1519–1539, nov 2006. → pages 16 and 94

[SWLH06] Andrew M Sutton, Darrell Whitley, Monte Lunacek, and Adele Howe. PSO and Multi-funnel Landscapes: How Cooperation might Limit Exploration. In *GECCO 2006: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 75–82. ACM, 2006. → page 38

[SYL13] Daniel L. Silver, Qiang Yang, and Lianghao Li. Lifelong Machine Learning Systems : Beyond Learning Algorithms. In *AAAI Spring Symposium Series*, volume 13, pages 49–55, 2013. → page 54

[Tai91] Eric D Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991. → pages 16, 96, 98, and 105

[Tal09] El Ghazali Talbi. *Metaheuristics: From Design to Implementation*. Wiley, 2009. → pages 17, 20, 21, and 22

[TS95] David M Tate and Alice E Smith. A genetic approach to the quadratic assignment problem. *Computers & Operations Research*, 22(1):73–83, 1995. → page 134

[TV14] Paolo Toth and Daniele Vigo. *Vehicle routing*. SIAM, 2014. → page 1

[VAB+13] S. Vonolfen, M. Affenzeller, A. Beham, E. Lengauer, and S. Wagner. Simulation-based evolution of resupply and routing policies in rich vendor-managed inventory scenarios. *Central European Journal of Operations Research*, 21(2), 2013. → page 1

[VABW11] S. Vonolfen, M. Affenzeller, A. Beham, and S. Wagner. Solving large-scale vehicle routing problem instances using an island-model offspring selection genetic algorithm. In *LINDI 2011 - 3rd IEEE International Symposium on Logistics and Industrial Informatics, Proceedings*, 2011. → page 18

[VBA+12] S. Vonolfen, A. Beham, M. Affenzeller, S. Wagner, and A. Mayr. *Combination and comparison of different genetic encodings for the vehicle routing problem*, volume 6927 LNCS. 2012. → page 1

[VBK+13] S Vonolfen, A Beham, M Kofler, M Affenzeller, and K Dörner. Simulation-Based Optimization of Transport Activities Within Cold Charge Steel Production. In *Proceedings of the 5th IEEE International Symposium on Logistics and Industrial Informatics (LINDI 2013)*, pages 67–73, Wildau, Germany, sep 2013. → page 1

[VCGP14] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673, 2014. → pages 1 and 18

[VDOT12] Sébastien Vérel, Fabio Daolio, Gabriela Ochoa, and Marco Tomassini. Local Optima Networks with Escape Edges. In Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, and Marc Schoenauer, editors, *Artificial Evolution*, pages 49–60, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. → page 45

[VFD05] S. Voß, A. Fink, and C. Duin. Looking Ahead with the Pilot Method. *Annals of Operations Research*, 136:285–302, 2005. → page 5

[VFM00] Vesselin K Vassilev, Terence C Fogarty, and Julian F Miller. Information characteristics and the structure of landscapes. *Evolutionary computation*, 8(1):31–60, 2000. → pages 35, 36, 73, 83, and 126

[VFM03] Vesselin K Vassilev, Terence C Fogarty, and Julian F Miller. Smoothness, ruggedness and neutrality of fitness landscapes:

from theory to application. *Advances in evolutionary computing*, pages 3–44, 2003. → page 34

[VW07] Stefan Voß and Andreas Witt. Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: A real-world application. *International Journal of Production Economics*, 105(2):445–458, 2007. → page 2

[Wag09] Stefan Wagner. *Heuristic Optimization Software Systems - {M}odeling of Heuristic Optimization Algorithms in the {HeuristicLab} Software Environment*. PhD thesis, Johannes Kepler University, Linz, Austria, 2009. → page 6

[Wey12] Dennis Weyland. A Rigorous Analysis of the Harmony Search Algorithm: How the Research Community can be Misled by a "Novel" Methodology. In Peng-Yeng Yin, editor, *Modeling, Analysis, and Applications in Metaheuristic Computing: Advancements and Trends: Advancements and Trends*, pages 72–83. IGI Global, 2012. → page 5

[WKB+14] S. Wagner, G. Kronberger, A. Beham, M. Kommenda, A. Scheibenpflug, E. Pitzer, S. Vonolfen, M. Kofler, S. Winkler, V. Dorfer, and M. Affenzeller. Architecture and Design of the HeuristicLab Optimization Environment. In *Advanced Methods and Applications in Computational Intelligence*, pages 197–261. Springer International Publishing, 2014. → pages 6 and 51

[WLM+18] Markus Wagner, Marius Lindauer, Mustafa M\is\ir, Samadhi Nallaperuma, and Frank Hutter. A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics*, 24(3):295–320, jun 2018. → page 54

[WM97] David H Wolpert and William G Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. → page 3

[Wri32] Sewall Wright. The roles of mutation, inbreeding, cross-breeding and selection in environment, 1932. → page 93

[WS11a] H Wang and M Song. Ckmeans.1d.dp: optimal k-means clustering in one dimension by dynamic programming. *The R Journal*, 3(2):29–33, 2011. → pages 111 and 143

[WS11b] John Robert Woodward and Jerry Swan. Automatically designing selection heuristics. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation -*

*GECCO '11*, GECCO '11, pages 583–590, New York, NY, USA, 2011. ACM. → page 53

[WSH08] Darrell Whitley, Andrew M Sutton, and Adele E Howe. Understanding elementary landscapes. In Maarten Keijzer, editor, *Proceedings of the 10th annual conference on Genetic and Evolutionary Computation*, pages 585–592. ACM, 2008. → page 42

[WZ99] A H Wright and Y Zhao. Markov Chain Models of Genetic Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 734–741, 1999. → page 21

[XHHLb08] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-brown. SATzilla: Portfolio-based Algorithm Selection for SAT. *J. Artif. Intell. Res.(JAIR)*, 32:565–606, 2008. → pages 52 and 158

[XHLb10] Lin Xu, Holger H Hoos, and Kevin Leyton-brown. Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010).*, pages 210–216, 2010. → pages 52, 53, and 158

[XY06] Yong Xia and Ya-Xiang Yuan. A new linearization method for quadratic assignment problems. *Optimisation Methods and Software*, 21(5):805–818, 2006. → page 16

[ZBRM13] Huizhen Zhang, Cesar Beltran-Royo, and Liang Ma. Solving the quadratic assignment problem by means of general purpose mixed integer linear programming solvers. *Annals of Operations Research*, 207(1):261–278, 2013. → page 16

[Zil96] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73–83, 1996. → page 17

[ZLT01] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Institute of Technology (ETH), Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001. → page 5

# Appendices

Algorithm 18 shows the description of the swap2 best-improvement local search for the quadratic assignment problem as used in Section 4.1.2. It uses an efficient calculation of the neighbors fitness and greedily selects the best-improving neighbor.

**Algorithm 18** Local Search in the Swap2 Neighborhood

```
 1: procedure Swap2Localsearch(↓ N, ↓ sol)
 2:     qual ← Fitness(sol)
 3:     isLocalOpt ← False
 4:     while isLocalOpt = False do
 5:         best ← (−1, −1)
 6:         bestDelta ← 0
 7:         for i ∈ [1; N) do
 8:             for j ∈ (i; N] do
 9:                 delta ← DeltaFitnessSwap(sol, i, j)          ▷ Fitness difference new - old
10:                 if delta < bestDelta then      ▷ Improvement in a minimization problem
11:                     best ← (i, j)
12:                     bestDelta ← delta
13:                 end if
14:             end for
15:         end for
16:         if bestDelta = 0 then
17:             isLocalOpt ← True                                  ▷ No more improving move
18:         else
19:             (i, j) ← best
20:             Swap(sol, i, j)
21:             qual ← qual + bestDelta
22:         end if
23:     end while
24:     return sol
25: end procedure
```

Algorithm 19 shows the `DiversitySelect` method used within Algorithm 9 in Section 4.1.2 on page 103 in a pseudo-code description. The algorithm creates a new population from the combination of the old population and the generated offspring by taking the best non-identical solutions.

**Algorithm 19** DiversitySelect Method for GLS

```
1: procedure DiversitySelect(↓ pop, ↓ offspring)
2:     len ← Length(pop)
3:     (s_1, s_2, ..., s_n) ← Sort(Concat(pop, offspring), by = Fitness, order = bestFirst)
4:     nextgen ← [s_1]
5:     similarset ← {s_1}
6:     for i ∈ [2; n] do
7:         if Length(nextgen) = len then
8:             break
9:         end if
10:        if s_i ∉ similarset then
11:            nextgen ←+ s_i
12:            similarset ← similarset ∪ s_i
13:        end if
14:    end for
15:    return nextgen
16: end procedure
```