THE WINNER DETERMINATION PROBLEM IN MULTI-UNIT COMBINATORIAL AUCTIONS AND THE MULTI-DIMENSIONAL KNAPSACK PROBLEM
AN ANALYSIS OF OPTIMIZATION METHODS

Diplomarbeit

eingereicht im: Oktober 2006
von: Jella Pfeiffer
geboren am 22. September 1981
in Ludwigshafen

Matrikelnummer: 0908058
## Contents

*Keywords* iv  
*List of Figures* v  
*List of Tables* vi  
*Abbreviations* vii  

1 Introduction 1  

2 Combinatorial Auctions 3  
  2.1 Motivation 3  
  2.2 Mechanism Design 4  
    2.2.1 Definition 4  
    2.2.2 Revelation Principle and Gibbard-Satterthwaite Theorem 5  
    2.2.3 Vickrey-Clarke-Grooves Mechanism 7  
  2.3 Auctions 8  
    2.3.1 Auction Design 8  
    2.3.2 Single-Good Auctions 9  
    2.3.3 Combinatorial Auctions 10  
  2.4 Winner Determination Problem (WDP) 11  
    2.4.1 Computational Aspects 11  
    2.4.2 Integer Program Formulation 13  
    2.4.3 Equivalence to the Multi-Dimensional Knapsack Problem (MDKP) 15  

3 Optimization Algorithms for the WDP and the MDKP 18  
  3.1 Algorithms for the Winner Determination Problem 18  
    3.1.1 Optimal Algorithms 18  
    3.1.2 Non-Optimal Algorithms 21  
  3.2 Algorithms for the Multi-Dimensional Knapsack Problem 22  
    3.2.1 Optimal Algorithms 22  
    3.2.2 Non-Optimal Algorithms 23  

4 Genetic Algorithms for the WDP and the MDKP 25  
  4.1 Functionality of Genetic Algorithms 25  
    4.1.1 Representation 27  
    4.1.2 Operators 28  
    4.1.3 Designing a Genetic Algorithm 31  
  4.2 Search Space of the WDP and the MDKP 34  
  4.3 Problem-Specific Bias by Bid-Sorting Heuristics 35
## Contents

4.4 Existing Genetic Algorithms for the MDKP ........................................ 38  
4.4.1 Direct Representation ...................................................... 38  
4.4.2 Indirect Representation ................................................... 41  
4.4.3 Comparison ................................................................. 47  

5 Analysis ............................................................................. 49  
5.1 Test Instances ............................................................... 49  
5.2 Performance .................................................................. 57  
5.2.1 CAMUS .......................................................................... 57  
5.2.2 CPLEX ........................................................................... 59  
5.2.3 Primal Greedy Heuristics ................................................. 61  
5.2.4 Raidl’s Weight-Coded Approach ........................................ 68  
5.3 Phenotypic Distance of Raidl’s Approach .............................. 70  
5.3.1 Expected Distance of two Individuals ................................. 71  
5.3.2 Distance to Primal Greedy Heuristics and CPLEX ............... 74  
5.3.3 Distance during Search Process ......................................... 76  
5.4 Phenotypic Duplicates of Raidl’s Approach ............................ 83  
5.5 Improving the State of the Art ............................................. 87  

6 Summary, Conclusions, and Further Work ............................... 91  
6.1 Summary .......................................................................... 91  
6.2 Conclusions ..................................................................... 95  
6.3 Further Work ..................................................................... 97  

A Appendix ........................................................................... 99
Keywords

Duplicate Elimination
Genetic Algorithm
Greedy Heuristic
Integer Program
Meta-Heuristic
Multi-Dimensional Knapsack Problem
Multi-Unit Combinatorial Auction
Phenotypic Distance
Winner Determination Problem
List of Figures

4.1 Functionality of a genetic algorithm ........................................ 26
4.2 Example of a genotype with binary alleles ............................... 27
4.3 Crossover operators for bit strings ........................................... 30
4.4 Search space illustrated as graph ............................................. 35
4.5 Direct representation with repair mechanism and local optimization . 40
4.6 Indirect representation with decoder ......................................... 41
4.7 Sensibility of LOG and LOGNORM distribution (1) ....................... 45
4.8 Sensibility of LOG and LOGNORM distribution (2) ....................... 47
5.1 Influence of input sizes on performance of primal greedy heuristics ...... 63
5.2 Average distance in populations with and without duplicates ........ 77
5.3 Minimal distance of children with and without duplicates ............. 79
5.4 Minimal distance of children with different tightness ratios ............. 82
5.5 Expected average distance ..................................................... 83
A.1 Minimal distance of children during the whole search process ........... 100
List of Tables

4.1 Comparison of different perturbation methods ......................... 46
4.2 Performance of best genetic algorithms for the MDKP ............... 48

5.2 Performance of CAMUS ..................................... 57
5.3 Performance of CPLEX ..................................... 59
5.4 Performance of primal greedy heuristic ............................. 62
5.5 Performance of Raidl’s approach ................................ 69
5.6 Expected average distance between two individuals ................. 73
5.7 Comparison between Hamming distances of solutions ................. 74
5.8 Comparison of runs with and without duplicates ..................... 78
5.9 Distance of best individual to population ........................... 80
5.10 Influence of perturbation size on produced duplicates ............. 84
5.11 Influence of perturbation size on in total produced duplicates ..... 86
5.12 Influence of mutation rate on produced duplicates ................ 87
5.13 Using different heuristic biases for the GA ....................... 88
5.14 Number of evaluations for different heuristic biases ............... 90

A.1 Optima for the MKNAPCB test instances ........................... 99
Abbreviations

BG           bid graph
BGG          bid-good graph
CA           combinatorial auction
DE           duplicate elimination
FPTAS        fully polynomial time approximation scheme
GA           genetic algorithm
KP           knapsack problem
MDKP         multi-dimensional knapsack problem
MUCA         multi-unit combinatorial auction
NDE          no duplicate elimination
PTAS         polynomial time approximation scheme
VCG          Vickrey-Clarke-Grooves
WDP          winner determination problem
1 Introduction

The increasing number of research fields in computer science evokes the danger of ignoring coherences between different domains. Similarities or equivalences of problems often remain unnoticed, wherefore the same questions are answered repeatedly, which might lead to a waste of human resources. This situation was as well true for the domains of combinatorial auctions and knapsack problems. Only recently, researchers have alluded to the equivalent integer program formulation of the multi-unit winner determination problem in combinatorial auctions and the multi-dimensional knapsack problem (Holte, 2001, Kelly, 2004). No work has addressed the issue of comparing solutions from both fields in detail ever since. Thus, this work will provide the missing integrative step, aiming at a more intense understanding and a mutual inspiration of both research areas. We will examine structural differences of test instances from both domains, compare the performance of different algorithms, and allude to a successful search behavior of meta-heuristics.

With the increasing interest in e-business since the 1990s, research in combinatorial auctions has gained a lot of importance. In combinatorial auctions agents are permitted to bid on all subsets of offered goods referred to as bundles. This opportunity allows them to express synergies between the goods they want to obtain, since the price they offer for one good might depend on other goods they will receive. The general case of combinatorial auctions includes multiple identical units per good, which is referred to as multi-unit combinatorial auctions. Once all agents have submitted their bids, the so-called winner determination problem must be solved. It describes the task of finding the optimal allocation of goods. Inevitably, allowing bidders to bid on all subsets of \( n \) goods makes the problem exponential in its problem size. Consequently, although such combinatorial auctions were proposed as early as 1976 for radio spectrum rights, only the advances in computer power has made it possible to start applying those mechanisms (Jackson, 1976, de Vries and Vohra, 2003).

A first algorithm for finding the optimal allocation of a single-unit combinatorial auction was published by Rassenti et al. (1982). They considered the context of selling time-slots at airports in order to permit airlines to bid simultaneously for take-off and landing time-slots. Current algorithms mostly address the special case of single-unit combinatorial auctions, in which only one unit per good is available (Hoos and Boutilier, 2000, Sakurai et al., 2000, Zurel and Nisan, 2001, Guo et al., 2005, Sandholm et al., 2005). Whereas only few researchers have addressed the more general case of multi-unit combinatorial auctions, in which only one unit per good is available (Hoos and Boutilier, 2000, Sakurai et al., 2000, Zurel and Nisan, 2001, Guo et al., 2005, Sandholm et al., 2005). Whereas only few researchers have addressed the more general case of multi-unit combinatorial auctions, in which only one unit per good is available (Leyton-Brown et al., 2000, Holte, 2001, Gonen and Lehmann, 2002). Most of them have tried to solve the problem to optimality. However, due to its sophisticated development in the meantime, the commercial software ILOG CPLEX outperforms all these existing optimal algorithms for the multi-unit case with respect to running time. Concerning the specialized single-unit case, only CABOB performs comparably well (Sandholm et al., 2005). We argue that literature in combinatorial auctions has been focussing on instances solvable to optimality, but leaving out relevant real-world scenarios. We will discuss that the problem difficulty of real-world scenarios can make optimal algorithms impractical for use due to a limit of
time and computational power. Wherefore the application of non-optimal algorithms is required. In the domain of multi-dimensional knapsack problems, non-optimal algorithms have been paid special attention to for a long time. This is due to standard test instances used in knapsack problems being much harder to solve than typical instances from research on combinatorial auctions. Since those harder instances have properties similar to the ones demanded for in combinatorial auctions, the algorithms solving them are interesting for both domains. Specifically meta-heuristics have shown to be very successful and should therefore be focussed on (Glover and Kochenberger, 1996, Chu and Beasley, 1998, Raidl, 1999).

In this work, various existing approaches for the multi-unit winner determination problem and the multi-dimensional knapsack problem will be examined. The goal is to investigate to which extend common ideas have been developed and in which context both domains can still learn from each other. We believe the basis for this analysis is to inspect the test instances used in both research areas. Common structures of instances must be found and their influence on the performance of different algorithms must be analyzed. This analysis will help determine under which conditions optimal algorithms fail. Afterwards, we can address the topic of non-optimal algorithms. We will focus on the analysis of one of the most successful meta-heuristics for the multi-dimensional knapsack problem, the weight-coded genetic algorithm by Raidl (1999). By presenting different experiments to observe its search behavior, we will be able to provide a deeper understanding of how the search space can be explored successfully. As a result, suggestions for improving the algorithm can be made.

This work is structured as follows. In chapter 2, some basics about combinatorial auctions will be pointed out. We will firstly introduce to mechanism design and, particularly, combinatorial auctions. Secondly, the IP formulation of the winner determination problem and its equivalence to the multi-dimensional knapsack problem will be shown. Afterwards, in chapter 3, a literature overview about various existing optimization methods for solving the problems will be given, while distinguishing between optimal and non-optimal algorithms. As an example of successful meta-heuristics applied for the multi-dimensional knapsack problem, a detailed examination of genetic algorithms will follow in chapter 4. First, an introduction to basic principles of genetic algorithms will be provided. Second, problem-specific knowledge about the search space and heuristic biases will be worked out. Finally, existing algorithms will be presented and evaluated. All three chapters lay the ground for the main chapter of this work, chapter 5, with the analyses. The analyses consist of four parts. First, the test instances are surveyed in respect of underlying structural properties. Second, the influence of structural properties on the performance of different algorithms is tested. After this, the weight-coded approach is scrutinized by measuring the phenotypic distance between individuals and the rate of produced duplicates. The observations will explain how the trade-off between exploration and exploitation is addressed in Raidl’s genetic algorithm. Finally, statements about potential improvements of this algorithm will be made. The work will close with a summary, a conclusion, and suggestions for further work.
2 Combinatorial Auctions

Combinatorial auctions (CAs) are a part of electronic market design. Research in electronic market design joins two disciplines: economics and computer science. Economical research focuses on game theoretical aspects by analyzing strategic behavior of self-interested agents. From the viewpoint of computer science, computational problems are addressed, such as finding the optimal allocation in auctions. As this work concentrates on computational aspects, we assume the reader has a stronger background in computer science than in economics. Thus, in this chapter we will point out the main ideas of the economical perspective to provide some basic knowledge in this area.

First, we will explain our motivation for researching in CAs. Afterwards, some basic concepts and results in the general field of mechanism design will be provided. Going from the general to the specific, we will then focus on the subfield auctions and, in specific, CAs. Finally, we will introduce in detail one of the major problems of CAs, the winner determination problem (WDP), which is in the focus of our work, and we will show its equivalence to the multi-dimensional knapsack problem (MDKP).

2.1 Motivation

With the rapid diffusion of the internet, different kinds of electronic commerce have emerged. Among those are auctions, which are one of the most popular game-theoretic mechanisms. Therefore, efficient auction design has become a subject of importance. Research on game-theory and multi-agent systems has started to examine which designs lead to preferable properties, such as the social welfare or truthful bidding. However, mechanisms are often used which put the bidders in the position of not being able to express their real valuations. One famous example is the single-good auction in which goods are offered independently from one another, applied, for example, at eBay Inc.\footnote{www.ebay.com}. In fact, if there are correlations between goods, much more efficient results could be achieved by using auctions that allow bidding on whole bundles of goods, so called CAs.

The advantage of CAs can be easily exemplified. Imagine an auction in which a travel agency offers a ship-cruise at the Rhine from Cologne to Mannheim with a one night stay-over at a hotel in Mannheim. At the same time it sells cheap tickets for the national theater in Mannheim for different performances of the same show. Being able to bid on the bundle \{ship-cruise, theater ticket\} for the same day, a bidder might take advantage of synergies since she could combine both events on the same date. As a result, she would value the whole bundle higher than the sum of the valuations for each good separately. In this case the utility function of the bidder comprises complementarities. On the other hand, think of an agent who wants to buy some shirts. Bidding separately on several shirts and hoping to receive at least one or two, the agent would risk ending up with an abundance of shirts. As the bidder’s value for all shirts together would be less than the
sum of all individual shirts, this utility function is said to have substitution effects.

To sum up, in CAs, the bidders have a lot more possibilities to express their valuations than in single-good auctions. They can consider synergies in their bids. With this game-theoretic advantage of CAs, some major disadvantages concerning the high computational complexity of the problem have to be seen. The drawback having received most attention in research is the WDP. This problem of finding an optimal allocation of goods is very complex. It is the subject of our work. Other problems such as communication complexity or preference elicitation are only explained briefly, since they are not focus of our analysis. In order to study the WDP more profoundly, it is necessary to gain some background knowledge in mechanism design theory and CAs.

2.2 Mechanism Design

2.2.1 Definition

Mechanism design was introduced by Hurwicz (1960). It aims at implementing system-wide solutions to problems in non-cooperative environments with multiple self-interested agents. Such problems can be political elections, public projects in which the participants themselves have to invest money, or allocation problems. Given that agents hold only private information about their preferences, a structure has to be chosen in which in equilibrium each agent behaves as the designer or principal intends. The designer can either act on behalf of the society, for example when collecting tax for a public project, or she can pursue self-interests when, for instance, being an auctioneer.

Since the agents’ information is private, the principal faces the problem that the agents might lie about their real valuations in order to influence the outcome according to their preferences. In most cases, whenever such manipulations occur, they damage the resulting system-wide welfare (Nisan and Ronen, 2000). Thus, simply asking the participants to reveal their preferences is unfavorable. Therefore, the principal has to define other rules which lead to the desired outcome. The most common solution to this problem is to introduce monetary transfers providing incentives for the agents to behave truthfully.

In mechanism design two economic areas are joined: game theory and social choice theory. In game theory the agents’ strategies are analyzed and in social choice theory an outcome is selected given a set of agents’ preferences. The outcome in social choice theory is determined by a social choice function, which is to be implemented by a mechanism. Formally we have a set of possible outcomes \( \mathcal{O} \) and agents \( i \in I, |I| = n \). Each agent \( i \) has a type \( \theta_i \in \Theta_i \) reflecting the possible preference orders the agent can have. The type captures all of the agent’s private information relevant to her decision. The agent’s utility \( u_i(o, \theta_i) \) over each outcome depends on her type; while \( u_i(o_1, \theta_i) > u_i(o_2, \theta_i) \) means that the outcome \( o_1 \) is preferred over the outcome \( o_2 \). The social choice function maps from the space of all types \( \Theta \) to the space of all outcomes \( O \),

\[
f : \Theta_1 \times \Theta_2 \times ... \times \Theta_n \rightarrow \mathcal{O}.
\]

Examples for such social choice functions are political voting protocols in which a candidate or a party is chosen or allocation problems. The most common objective of a social choice function is the maximization of the social welfare, the so called allocative-efficiency.
2.2 Mechanism Design

(Parkes, 2001). It maximizes the sum of all utilities over all agents:

$$f(\theta) = \arg\max_{o \in O} \sum_{i \in I} u_i(o, \theta_i). \tag{2.2}$$

Another objective is individual rationality; the agent’s payoff is never less by participating in the mechanism than her payoff without participating. Additionally there is Pareto optimality. An outcome is Pareto optimal whenever none of the agents could perform better without causing another agent to perform worse than in the current situation.

So far, we have learned what a social choice function is, and what typical objectives for the choices of outcomes are. Now, a mechanism has to be found which implements a given social choice function with one or several of these objectives. For this purpose, the agents’ possible strategies have to be specified together with an outcome function based on these strategies. The mechanism should guarantee an implementation despite the self-interest of the agents (Parkes, 2001). Mathematically, a mechanism \(M\) is defined on the strategy spaces \(S_i\) of the agents:

$$M = ((S_1, \ldots, S_n), g(\cdot)) \tag{2.3}$$

where \(g\) is an outcome function and \(S_i\) denotes all strategies or actions an agent \(i\) is allowed to take. A mechanism implements a social choice function if there is an equilibrium strategy profile \(s^*(\cdot) = (s_1^*(\cdot), \ldots, s_n^*(\cdot))\) of the game induced by \(M\) such that

$$g(s_1^*(\theta_1), \ldots, s_n^*(\theta_n)) = f(\theta_1, \ldots, \theta_n), \quad \forall (\theta_1, \ldots, \theta_n) \in (\Theta_1, \ldots, \Theta_n), \tag{2.4}$$

where \(s_i^*(\theta_i)\) is the strategy agent \(i\) with type \(\theta_i\) plays in the equilibrium. Please note that the equilibrium concept is not specified in this definition. It could, for example, be a Nash equilibrium. In this case, given the other players \(j, j \neq i\), conform to the equilibrium strategies \(s_j^*(\theta_j)\), no other player \(i\) has an incentive to unilaterally deviate from her equilibrium strategy. Other examples are the dominant strategy or the Bayes-Nash strategy equilibrium. The dominant strategy equilibrium makes it easy for the agents since the optimal strategy for an agent is independent of all strategies the other agents could play. Thus, the agents do not need to speculate what other agents might behave like. Informally, we could say that the concept of dominant strategies ”removes game theory from the problem” (Parkes, 2001, p. 5). The Bayes-Nash equilibrium is similar to Nash equilibriums, but assumes that agents have incomplete information about the opponents’ types. Therefore, agents use probability functions to speculate about the other agents’ preferences (Osborne and Rubinstein, 1994).

### 2.2.2 Revelation Principle and Gibbard-Satterthwaite Theorem

In equation 2.3, we see that a mechanism defines the available strategies and the function for selecting an outcome. It is necessary that these strategies are kept simple so that they can be applied by the agents. The easiest strategies occur when choosing a direct mechanism asking the agents to report their types directly to the principal, \(S_i = \Theta_i\). Direct mechanisms lead to a centralization of the problem as agents report their types to
2.2 Mechanism Design

a center that determines the outcome and reports it back to the agents. On the contrary, when applying *indirect mechanisms* agents have to think about how to transform their type into a strategy and the latter is reported to the mechanism. In other words,

"the computation that go on within the mind of any bidder in the non-direct mechanism are shifted to become part of the mechanism in the direct mechanism". (McAfee and McMillan, 1987, p. 712)

When applying these direct mechanisms agents may still lie about their true types. Mechanisms which, in contrast, succeed in establishing an equilibrium in which all agents tell the truth, are called *incentive-compatible*. In this case, it is in the interest of all agents to report their true types, \( s^*_i(\theta_i) = \theta_i, \forall \theta_i \in \Theta_i \). Further, if telling the truth is a dominant strategy, the mechanism is called *strategy-proof*. As will be shown later on, this can be achieved by the *Vickrey-Clarke-Grooves* (VCG) mechanism.

We learned that the equilibrium strategy profile \( s^*(\cdot) \) does not determine the concept of equilibrium. Some equilibrium concept must be chosen and implemented with the mechanism. In the worst case, in order to find out if a certain social choice function can be implemented by a certain mechanism with, for example dominant strategies, one would have to consider all possible mechanisms. However, research on mechanism design led to the *revelation principle* as a solution to this. It states that for any mechanism, there is a direct, incentive-compatible mechanism with the same outcome (McAfee and McMillan, 1987). An intuitive explanation for this principle is as follows: the transformation from types into strategies which occurs in the agents’ minds in indirect mechanisms is used as a filter in the direct mechanism. That is, the direct mechanism first filters all reports of the agents and simulates the indirect mechanism with the filtered input. This principle is valid for the optimal mechanism as well. Thus, the search for a mechanism can be focused on direct mechanisms. Therefore, if no direct mechanism can implement a given social choice function, then no indirect mechanism will do so.

In contrast to the positive result of the revelation principle, there also exists a negative result, the *Gibbard-Satterthwaite* theorem. According to it, it is impossible to find a mechanism with certain positive characteristics. To understand the theorem, first note that a social choice function is *truthfully implementable* if and only if the dominant strategy is to reveal the truth. Furthermore, a social choice function \( f \) is *onto* if for each \( o \in O \) at least one element in \( \Theta \) exists so that \( f \) maps to \( o \). Finally, a social choice function \( f \) is *dictatorial* whenever there is a dictator \( j \) among the agents so that for all outcomes, \( o_j \) is strictly preferred to another outcome \( o_k \) whenever the dictator \( j \) strictly prefers \( o_j \) to \( o_k \). Obviously, this is an unwanted characteristic. It makes the Gibbard-Satterthwaite theorem impractical for real-life mechanisms since they allow for manipulation.

**Gibbard-Satterthwaite Theorem:** Given \( O \) is finite, \(|O| \geq 3\), and the social choice function \( f \) is onto, then \( f \) is truthfully implementable in dominant strategies if and only if \( f \) is dictatorial.

According to the theorem it is impossible to elicit the truth if dominant strategies exist. Despite this result, however, the theorem can be circumvented by placing restrictions on the agents’ preferences, as it is done in the VCG mechanism.
2.2 Mechanism Design

2.2.3 Vickrey-Clarke-Grooves Mechanism

The VCG mechanism combines the following important virtues by introducing a special payment scheme. First, it implements social choice functions in dominant strategies. Thus, agents do not have to speculate which strategies the other agents might play, and they do not need to waste resources on learning about their competitors’ strategies. Second, the mechanism does not have to make any assumptions about the information agents have on each other. And, third, the VCG mechanism is allocative-efficient (see equation 2.2), strategy-proof and non-dictatorial. In fact, under weak assumptions, the VCG mechanism is the only direct mechanism with dominant strategies, allocative-efficient outcomes, and zero payments by losing bidders (Green and Laffont, 1979, Holmstrom, 1979). For a detailed description of the advantages and disadvantages of the VCG mechanism please refer to Ausubel and Milgrom (2006).

The attentive reader might think that the results contradict to the Gibbard-Satterthwaite theorem. The VCG mechanism, however, achieves these properties only by relaxing the assumption of unrestricted preferences: it assumes quasi-linear preferences. Formally, it defines an outcome \( o \in O \) and transfer payments \( t_i, i = 1, \ldots, n \), by all agents \( i \) to the center. In an auction, for instance, the set \( O \) would include all possible allocations of the available goods (Parkes and Shneidman, 2004). The value to each agent is then dependent on the outcome \( o \) and her type \( \theta_i, v_i(o, \theta_i) \). The VCG mechanism restricts the preferences agents may have to a set of quasi-linear preferences. With quasi-linear preferences, the agents’ utility functions are quasi-linear in the transfers and the values:

\[
  u_i(o, \theta_i) = v_i(o, \theta_i) - t_i.
\]

The key to make a mechanism allocative-efficient and strategy-proof is the way how transfers are computed. For better convenience, we denote \( o^* = \text{argmax}_{o \in O} \sum_i v_i(o, \theta_i') \) to be the allocative-efficient outcome for the reported types \( \theta' \). The VCG-payments are then computed as follows:

\[
  t_i(\theta') = \sum_{j \neq i} v_j(o^{-i}, \theta_j') - \sum_{j \neq i} v_j(o^*, \theta_j').
\]  

(L2.5)

Literally, this means that for each agent \( i \) the mechanism computes her contribution to the maximal outcome by taking part. This contribution is determined by computing two sums, each adding up all valuations over all agents except agent \( i \). The first sum assumes the outcome \( o^{-i} \) if agent \( i \) would not have participated, and adds up the valuations for this situation. The second sum takes the optimal outcome under the real, given situation, namely all agents take part. Consequently, the transfer payments are negative in case the agent makes a positive contribution to the system. That is because in this case the first sum, assuming agent \( i \) does not participate, is less than the second sum, assuming agent \( i \) participates. Since each agent pays her offer plus the transfer payment, an agent pays less if the system benefits of her participation.

As example take the suggestion to build a new park bench. The price of building this bench is 250\( \text{€} \) and only if \( \sum_{i=1}^{n} v_i \geq 250\text{€} \), the bench is built. Say, we have three agents 1, 2, and 3 and they all value the park bench differently: \( v_1 = 50\text{€} \), \( v_2 = 100\text{€} \), \( v_3 = 200\text{€} \). The transfer payment for agent 1 would then be: \( t_1 = (100 + 200) - (100 + 200) = 0 \), since without her participation the bench would be built as well, so \( v_2(o^{-1}, \theta_2') = v_2(o^*, \theta_2') = 100 \) and \( v_3(o^{-1}, \theta_3') = v_3(o^*, \theta_3') = 200 \). The other transfer payments are \( t_2 = (200 + 50) - (200 + 50) = 0 \) and \( t_3 = (0) - (50 + 100) \). Agent 1 and 2 do not have any transfer payments.
since their offers do not contribute to the construction of the bench. Agent 3, in contrast, receives a 150€ transfer payment, so that in total her utility is the value of 200€ plus the transfer payment of 150€: \( u_3(o, \theta_3) = 200 - (-150) \).

The VCG mechanism can be used to design auctions in which truthful bidding is the dominant strategy of all agents. In the single-good case this type is called a Vickrey Auction. The latter type of auction and other famous ones are presented in the next section. The proof of strategy-proofness in VCG mechanisms can be found in Ausubel and Milgrom (2006).

2.3 Auctions

2.3.1 Auction Design

The first who analyzed auctions from a game-theoretic viewpoint was Vickrey (1961). Later on, in 1966, he won the Nobel Memorial Prize in Economics - which was also due to his work on auction theory. In general, any auction mechanism should specify three elements (Lehmann et al., 2002):

1. **Bidding rules**: the goods offered, on which goods bids are allowed, and the timing of the bids.
   - Examples:
     - single-good auctions vs. CAs
     - single-unit (only one copy per good) vs. multi-unit (multiple copies per good)
     - ascending prices, descending prices, one-shot, etc.

2. **Market clearing rules**: determine when the auction is finished and what prices have to be paid.
   - Examples:
     - one round auctions vs. iterative auctions
     - first price vs. second price (the winner only pays the second highest bid price)

3. **Information disclosure rules**: which agents receive which information about the bids.
   - Example:
     - open-cry (agents bid openly) vs. sealed-bid (bids are submitted simultaneously so that no agent knows about other agents’ bids)

The aim of an auction mechanism is usually to guarantee Pareto optimal outcomes, to achieve allocative-efficiency, or to maximize the auctioneer’s revenue. Unfortunately these objectives often contradict (Myerson, 1981). In auctions, however, that use the VCG mechanism, allocative-efficiency is achieved and Pareto optimality is implemented, because both objectives are equivalent under quasi-linear preferences.

In this work we concentrate on the objective of maximizing the auctioneer’s revenue, as we think that in real-world scenarios this goal is the most relevant one. Furthermore, as we will see later, we cannot assume the bidders to bid truthfully. This makes it impossible to compute the social welfare correctly since it is based on knowing the true valuations.
2.3 Auctions

2.3.2 Single-Good Auctions

Four different common types of these auctions exist:

- English auction: first price, ascending, open-cry
- Dutch auction: first price, descending, open-cry
- Vickrey auction: second price, one-shot, sealed-bid
- First price sealed-bid: first price, one-shot, sealed-bid

The best known type is the English auction, as it is used, for instance, at Sotheby’s. Whenever no agent is willing to submit a higher bid, the highest bidder pays the price she has bid. The Dutch auction is very similar with the difference that the auctioneer lowers the price until an agent takes the bid at the current price. It was named for its usage at the Dutch flower market. As the name Vickrey Auction already implies, this kind of auction uses the VCG mechanism. This can be easily verified by recalling the payment scheme of the VCG mechanism:

\[ t_i(\theta') = \sum_{j \neq i} v_j(o^{-i}, \theta'_j) - \sum_{j \neq i} v_j(o^*, \theta'_j). \]

The transfer payment corresponds to the price the winner has to pay. It is the maximal value achieved had the winner not taken part (being equivalent to the second highest price) minus the sum of the values all other agents have for the realized optimal outcome. This value is 0 as all other agents win nothing. Here, the dominant strategy of truthful bidding in VCG mechanism becomes intuitively clear. Since the winner never affects the price she pays and her bid only determines whether she wins or not, no agent has an incentive to lie about her true valuations. The fourth type of auction, the first price sealed-bid auction, is like the third one, except that the winner pays the price of the highest bid. This pricing scheme does no longer guarantee truthful bidding.

The famous single-good auctions at eBay Inc. cannot be classified clearly in one of the four types since eBay has introduced automatic proxy bidding. In a proxy bid, a bidder can submit a reservation price, she is willing to pay. The bid will raise automatically to the previous bid price plus a small increment until the reservation price is reached. If there was no fixed end time at eBay, this principle would be comparable to a Vickrey auction with truthfulness as a dominant strategy. But because of a fixed deadline, last-minute bidding, the so-called *sniping*, makes it hard to anticipate the strategies of other players. Some might not use the proxy bidding and might misunderstand the system as an English auction by continuously raising their bids (incremental bidders). Then, sniping might be a best-response strategy and not using proxy bidding. When using proxy bidding, incremental bidders have more time to boost the price by placing their own bids, while sniping takes advantage of the fixed deadline. Increasing bidders might be to slow at closing time to further express their valuations. In other words, not in all cases bidding early by using a bidding proxy is the best strategy and therefore truth telling is not a dominant strategy as it is in the one-shot Vickrey auctions.
2.3.3 Combinatorial Auctions

The auction types presented in the last section are commonly applied to the single-good case. Nevertheless, driven by the development of mechanisms for the auctioning of different spectrums in the UK and in the USA\(^2\), the focus in the realm of electronic markets lies on a different kind of auctions: CAs (McMillan, 1995, Schwind and Stockheim, 2004). The advantage of these auctions, in which bidding is allowed on bundles of goods, is that agents can express synergies they have between various goods.

The synergies can be expressed by two different kinds of utility functions; those with 1) \textit{sub-additivity} expressing substitution effects between goods; and those with 2) \textit{super-additivity} expressing complementarities between goods. A function \(f\) is locally sub-additive if for all disjoint sets \(S_1\) and \(S_2\) holds that \(f(S_1 \cup S_2) \leq f(S_1) + f(S_2)\). A function \(f\) is super-additive if for all disjoint sets \(S_1\) and \(S_2\) holds that \(f(S_1 \cup S_2) \geq f(S_1) + f(S_2)\) (Cramton et al., 2006). In other words, sub-additivity occurs when the bundle is worth less than the valuation for the single goods, and super-additivity exists when the bundle is worth more than the sum of the single goods. Auctions, in which the geographical position of the goods plays an important role are typical examples of super-additive utility functions. Take the auctions selling radio spectra or those selling London bus routes as exemplification. Here it can be of use to win spectra from adjacent states or, in the other case, those bus routes that are connected with each other in order to overcome longer distances. In the extreme case, the interdependence of the goods in the aspired bundle can be so strong that winning only a subset of the goods creates almost no value for the agent. In this case, in the single-good auction scenario, the high risk of ending up with a valueless subset often leads to very cautious bidding reflected in low prices for the single bids. This risk is created by incomplete information of the bidders about each others’ valuations. This lack of information makes estimating which single-good auctions will be won extremely hard. In literature this problem is called the \textit{exposure effect} (Milgrom, 2004, Holland and O’Sullivan, 2004). In contrast, the effect of sub-additivity occurs when multiple alternatives of goods fulfill the same purpose. Think of an agent searching for a way of protection against rain. An umbrella might fulfill this purpose as well as a raincoat, however purchasing both might not be necessary since both goods are sufficient individually. In summary, the advantages of CAs create a win-win situation, both for the bidder and the auctioneer. If bidders are not exposed to the mentioned risks, they can bid higher prices, leading to higher revenue for the auctioneer.

So far, CAs have already been realized in some real-world scenarios. For example, at the University of Chicago the students can state which classes they intend to take by placing bundled bids, and a CA mechanism allocates the available capacity of the courses to the students (Graves et al., 1993). In another application transport capacities were auctioned. With the help of CAs, Sears Logistic Services\(^3\) reduced their procurement cost by 13% (Ledyard et al., 2002). Furthermore the London Bus Transport Company has sold about 800 bus routes per year in auctions since 1995 by using CAs (Kellerer et al., 2004). A final example is the supply of school meals in Chile where the state spends around $180 million to provide meals for 1,300,000 students. In a large state auction the


\(^3\)See https://www.slslogistics.com/.
mechanism was changed to a CA and the state saved $40 million, enough money to feed an additional 300,000 children. Overall, the application of CAs in real-world scenarios is of special interest for logistics and transportation, procurement, e-finance, the allocation of public goods, and supply chain management with scheduling problems (Schwind and Stockheim, 2004). For recent electronic markets prototypes conducting CAs have been developed. For details on these prototypes the reader is referred to Wurman et al. (1998) and Sandholm (2002b).

CAs facilitate bidding on goods with synergies. Nevertheless, some general problems with auctions still remain. The most famous ones are bidder collusion, a lying auctioneer, and winner’s curse. In bidder collusion, the bidders can reduce the revenue of the seller by collusive agreements. A lying auctioneer refers to the problem that the auctioneer could, for example, overstate the second highest price in a Vickrey auction in order to increase her revenue, or she could use fake bidders. Furthermore, the winner’s curse can occur in auctions where the value of a good is entirely determined by all agents’ values for this good (common value auction). The agent who wins has automatically bidden too much, since the average of all bids should only equal the real value of the good. As these problems lead too far into economics their discussion is beyond the scope of this work. The interested reader is referred to Weiss (2000).

2.4 Winner Determination Problem (WDP)

If a mechanism is only designed according to game-theoretical objectives, the problem may arise that the mechanism is computationally intractable. Therefore, important game-theoretic aspects must often be relaxed to make the mechanism implementable, such as the abandonment of strategy-proofness.

In this section, we will point out where the computational problems in an auction lie while focussing on the allocation problem, the so-called WDP. We will show that the mathematical formulation of the latter is quite easy and can be seen as a general linear integer problem. Yet, as the problem is \( \text{NP} \)-complete and no polynomial time approximation scheme (PTAS) does exist, it is a challenge to solve the WDP efficiently.

2.4.1 Computational Aspects

Computational complexity in auction design occurs within two distinct levels. The first level deals with the computational problems the agents face. The second considers aspects concerning the infrastructure of the mechanism (Parkes, 2001).

A summary of the levels is given below.

1. Agent level
   a) Valuation complexity: the complexity an agent faces to determine her values of the available goods or bundles of goods.
   b) Strategic complexity: the complexity of finding the best strategy which often includes anticipating other agents’ preferences.

\[ \text{See } \text{http://www.cs.cmu.edu/People/amem/emediator/}. \]
2. Infrastructure Level

a) Communication complexity: created by the amount of communication between agents and the infrastructure necessary to compute an outcome.

b) Winner determination complexity: the complexity of computing the optimal allocation of offered goods.

As this work only considers CAs, we now describe the different levels of complexity on the basis of this type of auction. The first complexity to discuss is the valuation complexity. Having \( n \) goods to bid on, the agent must determine all valuations for each combination she has an interest in. In the worst case this means bidding on \( 2^n - 1 \) bundles when one unit per good is available. In the multi-unit case this number is even higher. Therefore, the valuation complexity is much higher in CAs than in single-good auctions.

The degree of strategic complexity from a game-strategic viewpoint depends on the equilibrium concept which is implemented. If each agent’s dominant strategy is to bid her true valuations, which is the case in VCG mechanisms, then no strategic complexity remains. Agents do not have to speculate about other agents’ types. Yet, having, for instance, a Nash-equilibrium concept, each agent has to have an expectation of the behavior of all other agents. Here, the complexity is much higher than in the dominant strategy case.

The communication complexity is closely related to the valuation complexity. In a one-shot auction all agents reveal their bids at once. Consequently, the communication and the valuation complexity are high, since the agents have to reveal all values for the bundles they are interested in. But imagine an iterative mechanism, which in several rounds asks the agents to bid on specified bundles. By doing so, the iterative mechanism tries to cope with a minimum of information for finding the optimal allocation. This protocol has two advantages, first, the agent does not have to determine her complete preference order in advance and, second, she might approve the fact of not having to reveal her complete type, as competitors may take advantage of this information. The problem of eliciting just enough information to be able to compute the optimal allocation is called preference elicitation and has been studied in several papers (Blum et al., 2004, Lahaie and Parkes, 2004, Nisan and Segal, 2006). Examples for iterative mechanisms considering the reluctance of agents to reveal all their information can be found in (Parkes, 2001, Anandalingam et al., 2002, Biswas and Narahari, 2003).

Another aspect determines as well the communication complexity of a mechanism: the language in which bids are communicated. Often, it is assumed that an agent is willing to win multiples of the submitted, atomic bids and that she will pay the sum of the prices for those bids. In this case, we assume the agent to be super-additive in submitted atomic bids and call the bidding language OR bidding language according to the logical meaning of OR (Nisan, 2006). Yet, with the OR bidding language not all kinds of valuations can be expressed, as we have learned that utility functions can also be sub-additive whenever goods substitute each other. For this case, the fully-expressive XOR bidding language has been introduced. In it bidders may require that one of their atomic bids is accepted at most. As first presented in Fujishima et al. (1999) by allowing dummy goods the OR language can be made fully expressive as well. Dummy goods are simple artificial goods enforcing XOR constraints. Whenever \( n \) bids are supposed to be mutually exclusive
(meaning that at most one is intended to be accepted), one dummy good has to be added to the pool of available goods and it has to be included into each bundle. In this way, only one bid can be accepted, as only one unit of each dummy good is available. Hence, in many algorithms an OR bidding language with dummy goods is assumed (Leyton-Brown et al., 2000, de Vries and Vohra, 2003). The same assumption holds for the formulation of the problem used in this work. The auctioneer does not care about exclusivity between bids, since we assume that dummy goods have already been included in the submitted bids.

The WDP is the complexity problem that received most attention by far (Rothkopf et al., 1998, Hoos and Boutilier, 2000, Leyton-Brown et al., 2000, Holte, 2001, Gonen and Lehmann, 2002, Sandholm et al., 2002, de Vries and Vohra, 2003, Sandholm et al., 2005, Lehmann et al., 2006). It is the problem of finding the optimal allocation of available goods. As optimization criteria the auctioneer’s revenue is taken. The revenue is maximized by maximizing the sum of all submitted prices for the winning bids. Under the assumption that bidders reveal their preferences truthfully and that payments equal the prices offered for the bundles, maximizing the auctioneer’s revenue would be equal to maximizing social welfare.

Whenever the VCG is the mechanism applied, the WDP becomes very complex since for determining the VCG payments the WDP must be solved once for all n agents and n times for n – 1 agents, removing each agent once from the computation. The WDP problem itself is NP-complete since it is reducible to weighted set packing (Rothkopf et al., 1998) and the stable set problem (Lehmann et al., 2006). In a VCG, this means solving n + 1 NP problems. In connection with another drawback of VCG mechanisms the NP-completeness of the WDP, makes the VCG impractical for applications: VCG mechanisms require the WDP to be solved optimally (Nisan and Ronen, 2000). But, as the WDP is NP-complete, this undertaking is extremely hard for a huge number of bids and goods.

Of the four complexity problems, this work only deals with the WDP as we consider it the basic problem of CAs. Usually, the auctioneer is the initiator of an auction. However, as long as from her viewpoint CAs cannot guarantee good revenue by allocating the goods as profitable as possible, the auctioneer has no incentive to organize one. Unfortunately, the just mentioned disadvantage of VCG mechanisms leads to an important limitation of this work on WDPs. Since we observe non-optimal solutions of the WDP, we cannot implement VCG mechanism and therefore we cannot assume truthful bidding as a dominant strategy.

2.4.2 Integer Program Formulation

The WDP can be formulated as an integer program (IP) in which the decision variables \(x_j\) take value 1 if bid \(j\) is accepted and 0 otherwise. First, the single-unit case is described in which only one copy of each good \(i\) is available. The single-unit case is the most widely considered type in the CA literature,

\(^5\)VCG payments, for instance, do not correspond to the prices offered by the agents.
2.4 Winner Determination Problem (WDP)

\[
\text{maximize} \quad f = \sum_{j=1}^{n} p_j x_j \quad (2.6)
\]

\[
\text{subject to} \quad \sum_{j=1}^{n} r_{ij} x_j \leq 1, \quad i = 1, ..., m \quad (2.7)
\]

with \( x_j \in \{0, 1\}, \quad j = 1, ..., n \)

\( p_j > 0, \quad r_{ij} \in \{0, 1\} \),

where \( m \) is the number of goods, \( n \) the number of bids, and \( p_j \) the price paid for bid \( j \). The variable \( r_{ij} \) is 1 if and only if good \( i \) is included in bid \( j \) and 0 otherwise. Therefore, constraint 2.7 ensures that each good can be allocated to a bid at most once. The inequality in constraint 2.7 reflects the fact that good \( i \) needs not to be allocated: the auctioneer can as well keep the good. Wherefore, each good is assumed to have a positive or no influence when added to a bundle of existing goods. Thus, any agent would accept an extra good in her bundle. This common assumption is called free-disposal and can mathematically be expressed as follows: \( S \subseteq S' \Rightarrow u_i(S, \theta_i) \leq u_i(S', \theta_i) \).

Note that we do not restrict one agent to only place one bid. Consequently we have \( n \) or less agents participating in the auction. Furthermore, the represented formulation does not consider the case in which bidders have sub-additive utility functions. It does not specify which combinations of bids are allocated to one single agent. In other words, a bidder is assumed to pay \( p_1 + p_2 \) for the bids 1 and 2 and not a price less than this sum. Yet, as discussed before, by introducing dummy bids, we can obviate the problem and state that the given IP specifies the WDP sufficiently.

The formulation as IP shows its equivalence to the well-studied set packing problem which is known to be \( NP \)-hard (Karp, 1972, Rothkopf et al., 1998). Replacing the inequality symbol in 2.7 by an equality symbol, in other words having an auction without free-disposal, the resulting problem would be equivalent to another well-known problem: the set partitioning problem. Examples for auctions without free-disposal can be found in Vries and Vohra (2003) and for a detailed description of different packing and covering problems, the reader is referred to Gottlieb (1999a).

Lehmann et al. (2006) showed that the WDP is not only \( NP \)-hard but even \( NP \)-complete. The remaining question is how well the optimal solution can be approximated with a polynomial time approximation algorithm. Different approximation schemes exist, among them polynomial time approximation scheme (PTAS) and fully polynomial time approximation scheme (FPTAS). A PTAS algorithm \( H \) takes as input an error \( \epsilon \in (0, 1) \) and guarantees a solution \( z^H \) in polynomial time for the problem with \( z^H \geq (1 - \epsilon)z^* \), where \( z^* \) is the optimal solution. Whenever the running time is, additionally, guaranteed to be polynomial in \( \frac{1}{\epsilon} \), the approximation scheme is a FPTAS. Bermann and Fujito (1999) showed that there is no PTAS for the maximum stable set problem on graphs and Lehmann et al. (2006) conclude that same holds for the WDP. They argue that the maximum stable set problem is reducible to the latter taking the number of bids and the number of goods as input sizes.

In fact single-unit CAs are a special case of the more general multi-unit CAs (MUCAs). In multi-unit auctions, there are several copies of the same type of good auctioned. In
literature the multi-unit type of auction is a lot less studied than the single-unit case, although it is more general. Lehmann et al. (2006) argue that, conceptually, it would be possible to model a single-unit auction with identical copies of a good, but conclude that, computationally, this approach would be less favorable since it makes the bid representation much more complex.

As an example consider a manufacturer auctioning 50 identical cellular phones and 40 identical TVs. In other words, 50 units of the good cellular phone and 40 units of the good TV are available. A retailer who wants to buy 30 cellular phones and 10 TVs would be indifferent between all bundles with this amount of cellular phones and TVs. She would like to place a single bid (price, \{30 cells, 10 TVs\}). Instead, in single-unit auctions, she has to bid on all \(^{50}_{30} \times ^{40}_{10}\) distinct bundles, which would complicate the bidding process.

The mathematical formulation of the multi-unit case is very similar to that of the single-unit case, as only the codomain of some variables in the main constraint is enlarged:

\[
\begin{align*}
\text{maximize} & \quad f = \sum_{j=1}^{n} p_j x_j \\
\text{subject to} & \quad \sum_{j=1}^{n} r_{ij} x_j \leq c_i, \quad i = 1, \ldots, m \\
\text{with} & \quad x_j \in \{0, 1\}, \quad j = 1, \ldots, n \\
& \quad p_j > 0, \quad r_{ij} \in \mathbb{N}_0, \quad c_i \in \mathbb{N}_0.
\end{align*}
\]

We have seen that the single-unit version of the WDP is equivalent to the set packing problem. The multi-unit version, as well, is equivalent to a famous problem, the MDKP. This equivalence will be discussed in detail in the next section.

As the multi-unit auction is only a more complex version of the single-unit case, it is obvious that the negative results of PTAS algorithms are also valid for the multi-unit case. Given that there has only been little research on MUCAs, we are not aware of any deeper examinations of approximation schemes. Therefore, knowing the equivalence to the MDKP, we will resort to the results in this area of research.

### 2.4.3 Equivalence to the Multi-Dimensional Knapsack Problem (MDKP)

Knapsack problems (KP) are widely studied in computer science as being an example of a simple non-trivial IP model with binary variables. This model is already \(NP\)-complete (Karp, 1972). In the standard version of the KP there is one knapsack which is supposed to be packed with a number of items. Each item has a value, and it consumes a certain amount of a given capacity, like volume or weight. Mathematically the problem can be formulated as follows:
2.4 Winner Determination Problem (WDP)

\[ \text{maximize} \quad f = \sum_{j=1}^{n} p_j x_j \]  \hspace{1cm} (2.10)

\[ \text{subject to} \quad \sum_{j=1}^{n} r_j x_j \leq c, \quad i = 1, \ldots, m \]  \hspace{1cm} (2.11)

with \( x_j \in \{0, 1\}, \quad j = 1, \ldots, n \)
\( p_j > 0, \quad c, r_j \in \mathbb{N}_0. \)

Given the mathematical formulation the close relation between the single-unit WDP and the KP becomes immediately clear. An item in the KP corresponds to a bid in the WDP and its value is its price \( p. \) However, the constraint differs slightly. Though there is only one constraint as in the single-unit case, we have additionally a capacity \( c \) which can take any positive integer value, and the resource consumption \( r_j \) which is not in \( \{0, 1\} \) but can be any other positive integer value. In an auction scenario this would correspond to the case where only one good is auctioned of which \( c \) units are available while \( r_j \) stands for the number of units desired in bid \( j. \)

As it is the case with CAs, many different versions of the KP exist. Among them only the MDKP is of interest to us. This problem is also known as d-dimensional knapsack problem, multi-constraint knapsack problem, or multiple knapsack problem. Its formulation as IP is the same as the one for the multi-unit WDP, see equation 2.8, with \( d \) constraints or goods where dummy goods are used to express XOR bids. A version of the KP where XOR constraints are directly reflected in the IP formulation is the multi-dimensional multiple-choice knapsack problem (Kelly, 2004).

The name multi-dimensional refers to the different dimensions of capacity which must be taken into account, such as weight and volume restrictions for truck loading space. Research on MDKP was originally motivated by budget-planning scenarios. Here, a subset of projects must be selected, each generating a profit, but as well consuming capacity, like manpower, machinery, and computing power. Each resource reflects one dimension or constraint.

Mainly two factors motivate the analysis of the MDKP as special case (Kellerer et al., 2004):

1. The constraint matrix of the MDKP is in general dense which makes the IP difficult to solve, and

2. a trivial solution \( x_j = 0, \forall j, \) exists, while for other IPs finding a feasible solution can be as hard as finding the optimum.

The first factor reflects a difference between the multi-unit WDP and the MDKP. While in packing problems most often all items will consume some share of all existing resources (capacity dimensions), bidders in auction scenarios may not desire to have bundles including units of all existing goods. Therefore, dense as well as sparse constraint matrices play an important role in CAs.

Concerning the computational complexity, much research has been done with regard to MDKPs. Those results can be translated directly to the computational complexity of
2.4 Winner Determination Problem (WDP)

WDP in MUCAs. Caprara et al. (2000) have published a PTAS for the MDKP which has a faster running time than previous PTAS (Oguz and Magazine, 1980, Frieze and Clarke, 1984). The presented PTAS are, however, only polynomial in the number of bids \( n \). This contrasts the assumptions Lehmann et al. (2006) made for their results, in which the number of goods and bids are considered as input sizes. The PTAS given in the paper by Caprara et al. is a \( 1/m+1 \) approximation algorithm\(^6\) and runs in \( O(n^{\lceil \frac{m}{1+m} \rceil - m}) \) units of time. The bound shows the pseudo-polynomial character of the approximation algorithm as it is exponential in \( m \). Yet, in the case of MUCAs, the auctioneer can determine in advance how many goods are to be auctioned. Thus, with the proposed PTAS the auctioneer can estimate in advance in which order of magnitude the algorithm’s running time will be. Then again, with the possibility to run huge auctions in the internet, a large amount of goods can be auctioned. This makes the proposed PTAS impractical for real-world applications. Furthermore, the approximation bound \( \frac{1}{m+1} \) is a very low bound. Much better results can be achieved with, for example, meta-heuristics. To sum up, there exists no PTAS for the MDKP and the multi-unit WDP when taking the number of goods and the number of bids as input sizes.

The close relation between KP and WDP was first discussed by Holte (Holte, 2001). Until then, most authors only considered the equivalence of the single-unit WDP to weighted set packing problems. The advantages of this discovery are that the KPs are easy to understand, can be simply exemplified graphically, are widely used on an industrial basis, and they have been studied very intensively in literature. Recently, Kelly (2004) has presented a comparison between the WDP for different kinds of auctions and their equivalent KPs. He states that research in WDP algorithms might profit a lot from the algorithms developed for the MDKP. So far, no real effort has been made in unifying methods of both research areas. However, as we will see later on, meta-heuristics have been successfully applied to the MDKP, and their adaptation and application to the multi-unit WDP seems a very promising direction.

\[^6\]z_H \geq (1 - \frac{1}{m+1})z^*
3 Optimization Algorithms for the WDP and the MDKP

This chapter will provide a literature overview of existing solutions to the WDP and the MDKP. We will point out that optimal algorithms for the WDP are already very sophisticated, whereas meta-heuristics for the multi-unit WDP are almost non-existent. Overall, for MUCAs only very few algorithms exist in literature. However, the ideas of the well-studied special case of single-unit auctions might be adopted for the general case. Therefore we will present those algorithms as well. Meta-heuristics, and specifically genetic algorithms (GAs), have demonstrated their success in the field of MDKP.

3.1 Algorithms for the Winner Determination Problem

Generally, there are two approaches to address the WDP: optimal and non-optimal algorithms which are each applied either to the original problem or a restricted version of the problem. Because of the NP-completeness of the WDP, the attempt to find optimal algorithms for the unrestricted problem is computationally only tractable for easy problem instances. Therefore, there are trials to restrict the problem to classes of sub-problems which can be solved optimally by polynomial time algorithms. Restrictions are put in place by forbidding certain combinations of bids, which narrows the agents in expressing their real valuations. Hence, the disadvantages of auctions like the exposure problem, and limited possibilities to express valuations, recur.

Concerning non-optimal approaches with restricted bids, algorithms were developed which provide incentive compatibility (Bartal et al., 2003). In this class of algorithms polynomial worst-case run-times can be guaranteed. Finally, non-optimal algorithms for the original problem usually cannot ensure any bound, but just try to be as accurate as possible. The advantages of those algorithms are their short running times and their applicability to real-world scenarios. The disadvantage of non-optimal algorithms for the WDP is that they cannot guarantee truthful bidding, as the VCG mechanism only works with optimal algorithms.

Note that usually the optimal solutions of test instances are unknown. Therefore, when applying non-optimal algorithms, the gapR to the relaxed LP solution \( f_{LP} \) is reported as benchmark. The gapR is defined as \( 1 - \frac{f}{f_{LP}} \), where \( f \) is the solution of the non-optimal algorithm.

3.1.1 Optimal Algorithms

Restricted Problem

Attempts to tackle the WDP have been made by disallowing bidding on certain bundles. This causes that bidders might not be able to bid on all bundles they desire. Research in that area tries to reach two goals. On the one hand, the restriction of bids must be designed in such a way that allows a polynomial algorithm to solve it optimally. On the other hand
the restricted problems must still be of practical relevance, in other words, scenarios must be found where the bidders are naturally not interested in certain combinations of goods.

The initial research in this area was done by Rothkopf et al. (1998). They considered different structures of permitted bids, gave examples of their relevance in real-world applications, and provided polynomial algorithms to solve the resulting WDP optimally. For instance, in cardinality-based structures, bids are only allowed to contain at most two goods. Allowing three goods makes the problems again not solvable in polynomial time. This restriction might be valid for auctions of airport landing slots since flight companies are mostly interested in pairs of landing and takeoff slots.

Other work in this kind of algorithms has concentrated on the mathematical properties of submitted bids. The bids determine the constraint matrices, which must have certain properties to be computational tractable (de Vries and Vohra, 2003). In this work, examples for practical relevance are hardly considered.

In Tennenholtz (2000) tractability for optimal algorithms is shown for the multi-unit version of cardinality-based structures, where agents are allowed to submit bids on two goods with a price $p$ and the quantities $l$ and $k$ for the desired goods respectively. Additionally, a proof for the tractability of multi-unit linear goods is shown. These are goods which have an order, like time slots in a schedule, and only neighbored parts, or intervals, can be acquired.

### Unrestricted Problem

In this class, algorithms for the WDP have concentrated on using branch and bound methods. Branch and bound methods construct decision trees, and try to eliminate certain regions of the solution space by pruning sub-trees. This is accomplished by finding lower and upper bounds for the unknown optimal solution. Other approaches, for instance, IDA* search and dynamic programming have been proposed (Rothkopf et al., 1998, Sandholm, 1999), and dynamic programming ideas have been incorporated in the branch and bound algorithm CAMUS, which is discussed later (Leyton-Brown et al., 2000). The first algorithm using branch and bound, CASS, was introduced in 1999 for the single-unit case. This algorithm has since been refined (Fujishima et al., 1999).

Due to its rapid development in the last years, the commercial solver ILOG CPLEX has become the state-of-the-art algorithm for the WDP. The first special algorithms for the single-unit WDP published in 1999 and 2000 were benchmarked against CPLEX 6.5 (Andersson et al., 2000, Fujishima et al., 1999, Sandholm, 1999). It was shown that CPLEX outperformed these algorithms. It ran about five magnitudes faster than Sandholm’s algorithm, and seemed faster than CASS, as well. Because of the sophisticated improvements in CPLEX routines since then$^1$, CPLEX is not only applied to the WDP, but to many other problems in operation research (Schuurmans et al., 2001).

Nevertheless, very good results of specialized algorithms for the single-unit WDP, using branch and bounds methods, have been found recently (Sandholm et al., 2001, Sandholm and Suri, 2003, Sandholm et al., 2005). In one of their latest publications Sandholm et al. (2005) present a sophisticated algorithm, CABOB, which branches on bids and uses many other techniques like decomposition, structural observations, bounding by using the

---

$^1$CPLEX 7.0 is about 2 times faster than CPLEX 6.5 (Sandholm et al., 2005).
3.1 Algorithms for the Winner Determination Problem

LP-solver of CPLEX, and sorting of bids with different heuristics. They compare CABOB with CPLEX and show that, in general, both algorithms take about the same running times. Actually, CABOB is faster on highly structured problems, due to its problem-specific knowledge of combinatorial auctions. Furthermore, they report that CABOB takes only linear space while CPLEX needs exponential space and runs out of virtual memory. Additionally, CABOB has a better anytime performance. In the paper by Sandholm and Suri (2003), it is remarked that CABOB can also be adapted to multi-unit problems, but no experiments are made, and in the latest version from 2005 multi-unit auctions are not considered at all.

For MUCAs two algorithms have been developed. Like for the single-unit case they both use branch and bound methods. The first algorithm is CAMUS (Leyton-Brown et al., 2000). It is a generalization and extension of CASS (Fujishima et al., 1999). It uses branch and bound with depth first search, pre-processing techniques, dynamic programming, several heuristics for search orderings, and some structural properties. The algorithm is tested on instances with 10 and 14 goods with 40 units in total and up to 2500 bids. The largest problems are solved optimally in two minutes with a good any-time performance. The authors show experimentally that although CAMUS remains sub-exponential in time in the number of bids, it remains exponential in the number of goods. The second exact algorithm for multi-unit combinatorial auctions uses bounds by solving several relaxed linear problems (Gonen and Lehmann, 2002). The relaxation is achieved by allowing the decision variables to take continuous values between zero and one. It can then be solved in polynomial time by a linear program (de Vries and Vohra, 2003). Furthermore, heuristics are used to determine which bid to branch on next. The authors compare their algorithm to CAMUS, although their comparison is not very detailed, as they carry out no experiments with CAMUS itself. They state that their algorithm is as fast as CAMUS for larger problem instances with 14 and 16 goods. Additionally, they show experimentally that, in contrast to CAMUS, the running time stays sub-exponential in the number of goods by testing auctions with up to 120 goods. However, the tests are only performed using a small number of bids, namely 125. The amount of units per good auctioned is unknown, they probably take the same maximal amount of five units per good as done in the test sets for CAMUS. We think that it is worthwhile to further consider both algorithms, but a more detailed comparison between the two is necessary.

We have seen that for the single-unit case good results have been found that perform comparably to CPLEX. However, research in the multi-unit case is rare and further work has to be done with detailed comparison with the state-of-the art solutions, like CPLEX. Furthermore, the optimal algorithms for the multi-unit case only work well with relative small input sizes. Therefore, it is necessary to consider other classes of algorithms which can deal with a larger number of bids and goods. In our opinion, with the development of internet auctions, scenarios with several hundred of goods and bids will become more realistic.

Please note that many algorithms use some kind of heuristic to decide which part of the search tree to prune on next. These heuristics reflect problem-specific knowledge and will become very relevant for the non-optimal algorithms. A more detailed presentation of the heuristics will be provided in section 4.3.
3.1 Algorithms for the Winner Determination Problem

3.1.2 Non-Optimal Algorithms

Restricted Problem

The advantage of algorithms in this class is that they are very fast; unfortunately, at the expense of optimality and restriction of the permitted bids. An important and well-studied kind of restriction is to assume bidders to be single-minded, desiring a single subset of goods. For this kind of auctions polynomial-time algorithms which are even incentive compatible exist (Lehmann et al., 2002, Mu’alem and Nisan, 2002). A worst case bound of $\sqrt{m}$ to the optimal solution, where $m$ is the number of goods, is provided for the approximation. This bound is, up to a multiplicative constant, the best bound for a polynomial-time algorithm with single-minded bidders.

In the multi-unit case, the work by Bartal et al. (2003) focuses on auctions in which exactly $k$ units of each good exist, and each agent desires a bounded amount of units of each good. The mechanism presented is incentive compatible and guarantees an approximation bound.

Unrestricted Problem

Non-optimal algorithms are useful for real-world applications as they are fast without restricting the bids which can be submitted. For small input sizes optimal algorithms are evidently preferable as only with those optimal algorithms VCG payments can be applied (Nisan and Ronen, 1999, Lehmann et al., 2002). With larger problem instances, non-optimal algorithms must be applied.

One of the simplest methods is to use a greedy algorithm. It sorts the bids by a specified criterium and accepts bids according to that order, as long as no capacity restrictions are violated. Such a heuristic is called a construction heuristic. Imagine we have $m$ goods in the single-unit case. The bids $j$ could be sorted in descending order of $v_j = \frac{p_j}{\sum_{i=1}^{m} r_{ij}}$, which reflects the average price of one unit in the bid. If the acceptance of the bid with the highest value does not violate any capacity restrictions, the corresponding decision variable $x_j$ is set to 1. Afterwards, all other bids are considered in order and added as long as they do not conflict with any constraints. This greedy heuristic was first applied to the set packing problem, equivalent to the single-unit WDP, and later on, explicitly for combinatorial auctions (Fisher and Wolsey, 1982, Holte, 2001, Zurel and Nisan, 2001).

Zurel and Nisan (2001) succeeded in being up to 1000 times faster than CPLEX while obtaining solutions with a gapR no worse than 4%. They used approximations by solving the relaxed LP as well as several greedy heuristics.

Holte (2001) also considered different heuristics for sorting the bids, which will be presented in section 4.3. In his work he also tested some heuristics for MUCA. To our knowledge, these are the only relevant non-optimal algorithms presented for the multi-unit WDP. The results show that one out of the three tested heuristics achieve a gapR less than 2% for some standard test instances of the MDKP\(^2\).

Concerning other algorithms for the single-unit case, several meta-heuristics have been applied. Among them a stochastic local search technique which computes approximations

---

\(^2\)Holte was the first who pointed to the equivalence of the multi-unit WDP and the MDKP, which is why he took instances from the MDKP literature.
3.2 Algorithms for the Multi-Dimensional Knapsack Problem

Algorithms to solve the MDKP are classified in the same way as the ones for the WDP. Since the focus of this work is on meta-heuristics, the overview of optimal algorithms will be brief. Furthermore, a thorough investigation of GAs for the MDKP is given in the next chapter.

3.2.1 Optimal Algorithms

Restricted Problem

To our knowledge, not many restrictions have been considered in literature dealing with the general MDKP. The reason for this might be that restrictions are more relevant for applications of the MDKP, since, there, connections to real-world scenarios are more obvious.

Some algorithms in the area of MDKP examine problem instances that contain a small number of constraints such that the number of dimensions is small. Fréville and Plateau (1996) presented, for instance, an optimal polynomial-time algorithm for the 2-dimensional knapsack problem. In the combinatorial auction scenario this means that only 2 different goods are auctioned. To our knowledge, such restrictions have not been considered in the literature on WDP, although for multi-unit auctions the cases having only a small number of goods, but a large number of copies per good might be quite interesting.

Unrestricted Problem

Comparable to the WDP, optimal algorithms for the unrestricted MDKP mainly use branch and bound methods and dynamic programming. Recently, dynamic programming has received little attention due to its limited success (Gilmore and Gomory, 1966, Weingartner and Ness, 1967, Nemhauser and Vance, 1994). Algorithms based on branch and bound use Lagrangian, surrogate, or composite relaxation for upper bounding, similar to the exact algorithms for the WDP (Gavish and Pirkul, 1985, Osorio et al., 2004). In other words, for the MDKP the same approaches are used as for the WDP with the same tendency to concentrate mostly on branch and bound methods. For a more detailed
3.2 Algorithms for the Multi-Dimensional Knapsack Problem

overview of optimal algorithms, the reader is referred to Chu and Beasley (1998) and Kellerer et al. (2004).

3.2.2 Non-Optimal Algorithms

To our best knowledge, no non-optimal algorithms for restricted versions of the MDKP exist. Therefore, in this section we only focus on the unrestricted case.

Unrestricted Problem

Generally, literature on MDKP distinguishes two basic construction heuristics for building a solution. The first is called primal greedy heuristic (Toyoda, 1975). It starts by sorting all items according to some criteria by decreasing value. Then it adds them one after another to the knapsack as long as no restrictions are violated. This heuristic is equivalent to the construction heuristic, described in section 3.1.2. The second heuristic, called dual greedy heuristic, starts the other way around (Senju and Toyoda, 1968). It first sets all decision variables to one, sorts the items by increasing value according to some sorting-heuristic, and, finally, removes them one after the other as long as capacities are violated. In the last step all removed items must be considered again for inclusion since by removing items new capacity might have been freed. This post-processing step is performed with the descending order of the items, as the most valuable item should be included.

To demonstrate the functioning of the dual greedy heuristic, consider this small example. There are three items, which are assigned a value according to some heuristic and we have two capacity restrictions $c_1 = 3$ and $c_2 = 2$. Item 1 has value 10 and consumes 1 of $c_1$ and 1 of $c_2$. Item 2 has value 20 and consumes 1 of $c_1$ and 2 of $c_2$. Item 3 has value 30 and consumes 2 of $c_1$ and 1 of $c_2$. In the beginning, the dual greedy heuristic sets all three decision variables for the three items to 1. As all items together consume more than the available capacities, the item with the least value is removed: item 1. Now, the capacity consumption is still too high with 3 of $c_1$ and 3 of $c_2$. In the next step, the heuristic removes item 2, the next lowest value. The capacity consumption resuming is 2 of $c_1$ and 1 of $c_2$. Packing only item 3 in the knapsack is a valid solution. However there still is some free capacity of 1 of $c_1$ and 1 of $c_2$. That is why we need the post-processing step of adding again some unpacked items, this time in decreasing order. First, item 2 is considered. It cannot be packed as the capacity would overflow. Nevertheless, we can add item 1 and gain an additional value of 10. Therefore, the capacity restrictions are met precisely. In this case, the solution found by the greedy heuristic is optimal.

The success of the two greedy-heuristics depends on the sorting-heuristic (Dobson, 1982, Fox and Scudder, 1985). Different sorting-heuristics are presented in section 4.3. Among them are advanced heuristics which make use of the relaxed problem with Lagrangian and surrogate multipliers (Magazine and Oguz, 1984, Pirkul, 1987). Other heuristics apply several relaxation-based LPs by first setting all decision variables which are 1 in the solution of the relaxation to 1 in the binary solution. After that, new relaxed LPs are solved with the remaining variables being above a specified threshold (Bertsimas and Demir, 2002).

Research on meta-heuristics has addressed the topic of MDKP as well (Battiti and Tecchiolli, 1995). Tabu search has been applied (Hanafi and Freville, 1997, Glover and
Kochenberger, 1996) as well as ant colony systems (Fidanova, 2002). The ant colony systems, however, exhibit bad results compared to state of the art meta-heuristics. Most successful attempts have been made by work in evolutionary computation (Khuri et al., 1994, Chu and Beasley, 1998, Raidl, 1999, Gottlieb, 2001), and will be discussed in detail in the next chapter. Further promising results have been found by hybridizing different heuristics. One of the first algorithm in this area was presented by Thiel and Voss (1994), who presented a hybrid GA with tabu search. Other work tries to compute the surrogate multipliers which were presented by Pirkul (1987) with the help of a GA or Lagrangian multipliers (Alonso et al., 2005, Yoon et al., 2005). Recently, a successful approach has been made by hybridizing branch and bound and evolutionary algorithms (EA), letting both algorithms run in parallel with a mutual exchange of found solutions (Gallardo et al., 2005). The branch and bound method uses the solutions of the EA to prune the branches which have an upper bound that is smaller than the lower bound obtained by the EA. On the other side, the branch and bound algorithm communicates promising regions in the search space to the EA guaranteeing a focused search.

We would like to explicitly point out the boundaries of the algorithms we will concentrate on from now on. From the combinatorial auction perspective, we consider only one-shot sealed-bid auctions with discrete allocation. Discrete allocation refers to goods or copies of goods that cannot be separated in multiple parts as it would be the case with liquids, for instance. Furthermore, we do not care about truthful bidding and assume that all problems before the WDP, concerning preference elicitation and communication of bids are already solved in some way. Last but not least, we focus on multi-unit combinatorial auctions.
4 Genetic Algorithms for the WDP and the MDKP

In the last years there has been growing interest in heuristic search methods. Specifically GAs have become very popular. The same holds for the field of MDKP where a lot of approaches use heuristics. In particular GAs have shown to be very successful algorithms in this area. Relating to the WDP, almost no GA has been developed, neither for the single-unit nor for the multi-unit case. That is why, at the end of this chapter, we will concentrate on presenting GAs only from research on MDKP. Before that, we will describe the functionality of GAs. Readers who are already expert in this area can directly skip to the second part. There, we will provide some problem-specific knowledge, by analyzing the search space the algorithm explores. After that, we will point out some heuristics which are used to bias the search in the evolutionary process. As was already mentioned, the chapter will finish with an overview of existing GAs for the MDKP. Among them, the focus lies on the weight-coded approach which will be the subject of the analyses in the subsequent chapter.

4.1 Functionality of Genetic Algorithms

In the 1960s and 70s, researchers came up with the idea of using principles of biological evolution to solve optimization problems (Fogel et al., 1966, Rechenberg, 1965, Rechenberg, 1973, Schwefel, 1975). Later on, in his book *Adaptation on Natural and Artificial Systems* (1979) Holland presented his idea of GAs. His main goal was to bring natural adaptation into computer systems. Since then, this algorithmic concept has flourished and attracted a lot of interest. Due to the enhancements by several scientists, among them Goldberg (1989), they have been applied to many scientific and real-world problems, like routing and scheduling (Storer et al., 1995, Beasley et al., 2001), aerospace engineering (Sasaki et al., 2001, Williams et al., 2001), and financial markets (Mahfoud and Mani, 1996, Andreou et al., 2002).

GAs are methods for which no approximation bounds can be guaranteed a priori, as such they are applied to problems where exact methods fail and where approximation algorithms are too slow or provide bad results. Among the non-optimal algorithms, GAs can be classified as *meta-heuristics*, which means that they apply a general, domain and problem independent concept to control the search process. This global principle uses generations of individuals (populations) that evolve by being subject to three main operations: selection, crossover, and mutation. Central to this method are the individuals who represent an encoding of a potential solution. They consist of several alleles describing the solution, just as chromosomes in biology describe characteristics of an organism. At the start of a GA, an initial population, consisting of a specified number of individuals, must be created. In the following process new populations are generated by reproduction. According to Darwin’s principle of natural selection only the fittest individuals survive the selection process and are allowed to reproduce by recombination (Darwin, 1859). The offspring inherits properties of their parents; therefore parents and offspring are similar.
4.1 Functionality of Genetic Algorithms

The idea is that this method guarantees good individuals, in other words, good solutions to the problem, to breed in upcoming generations. Yet, small random changes in the offspring, so-called mutations, let new properties emerge. In contrast to real life, where evolution is an ongoing process, an algorithm must stop. Therefore a stopping criteria has to be defined, for example, a fixed number of generations. In figure 4.1 the functionality of a GA is illustrated.

Generally, we can distinguish between the overlapping and the non-overlapping generation model. In the non-overlapping GA the offspring replaces the parents’ generation completely. Examples which use this method are the generational GA and the \((\mu, \lambda)\) version of evolution strategies, where \(\mu\) signifies the number of parents and \(\lambda\) the number of offspring created in each reproduction phase. In contrast, in the overlapping version, the offspring competes with the parents in the selection process which increases the selection pressure. Approaches working with the overlapping scheme are so-called steady-state algorithms and the \((\mu + \lambda)\) version of evolution strategies. The "\(+\)" symbol signifies that the next generation is chosen out of the pool consisting of parents and children. A common choice for the number of offspring in a steady algorithm is one, since then only the size of \(\mu\) is a parameter to be optimized.

When using evolutionary concepts in algorithms, it is clear that it cannot succeed without any incorporation of prior knowledge about the problem. This notion is a consequence from the well-known No Free Lunch theorem. This theorem states that across all possible problems the average performance of two algorithms is identical (Wolpert and Macready, 1995). Later on, Schumacher (2000) and Schumacher et al. (2001) sharpened the theorem stating that it only holds under certain assumptions. The reader is referred to Schumacher et al. (2001) for more details about those assumptions. Ho and Pepyne explain the theorem as follows:

"...a general-purpose universal optimization strategy is theoretically impossible, and the only way one strategy can outperform another is if it is specialized to the specific problem under consideration." (Ho and Pepyne, 2002, p. 550)
4.1 Functionality of Genetic Algorithms

Problem-specific knowledge is reflected in the representation together with the operators. By defining how solutions are represented, and in which way the individuals produce new offspring for the next generation, the search process can be guided in a way to be most successful. For this purpose it is essential to think about the structure and design of the search space and the interplay between operators and the movements of individuals in the search space.

In this section we will first describe the representation of solutions, then, the three operators, and, finally, we will focus on guided search in the search space induced by those operators. The latter observation explains the general notion of the trade-off between exploration and exploitation, which is the key viewpoint of the later analysis. Design decisions concerning representations and operators cannot be made independent from another, hence we will discuss them in this last part.

4.1.1 Representation

A representation indicates the assignment of a string of symbols to all potential solutions for the problem under consideration. In other words, it describes the encoding of a solution as an individual. We could say that the language in which solutions are described in an optimization problem is translated into some kind of allele-language one can work with in a GA. Since research has shown that the success of GAs is strongly affected by their representation, much work has been published concerning the design of representations (Goldberg, 1989, Liepins and Vose, 1990, Radcliffe, 1991a, Radcliffe, 1991b, Palmer and Kershenbaum, 1995, Ronald, 1997). Later on, Rothlauf (2005) has developed a theory of representations which uses theoretic models to predict the influence of a representation on the performance of a GA.

To describe an individual one distinguishes between genotype and phenotype. In biology the genotype stands for the information stored in the chromosomes while the phenotype of an individual determines its outer appearance (Rothlauf, 2005, p. 10). GAs adopt this notion by distinguishing between the phenotype of an individual, on whose basis it can evaluate the quality or fitness of the individual, and the genotype, on which it can apply the operators recombination and mutation to create new offspring. As an example, take a group of alleles in the genotype, say 110, which provides phenotypic information about the eye-color (see fig. 4.2). A representation specifies the mapping from the genotype to the phenotype, so from 110 to the eye-color. In other words it determines which genes translate into which properties of the individual. In this way, it is possible to encode a large number of phenotypes with the help of a small number of alleles, ordered in some sequence in the genotype.

Figure 4.2: Example of a genotype with binary alleles. In most genetic algorithms the genotype consists of only one chromosome.
Formally, to evaluate the quality of an individual, a fitness function must be defined. It maps from the genotypic space to a fitness value:

$$ f(x) : \Phi_g \rightarrow \mathbb{R}. $$

The fitness value stands for the objective value of the optimization problem, represented by the allele-vector $x$ of the individual. Whenever it is distinguished between a genotype and a phenotype of an individual, this is called an indirect representation. Consequently, a direct mapping from the genotype space to the fitness value space is impossible. Therefore, we have to split the fitness function into two parts: $f_g$ and $f_p$. The first maps the genotype to the corresponding phenotype, and the second assigns the according fitness value to the phenotype:

$$ f_g(x^g) : \Phi_g \rightarrow \Phi_p, $$

$$ f_p(x^p) : \Phi_p \rightarrow \mathbb{R}. $$

In some cases a genotype can be chosen to be equivalent to the phenotype. This is advisable if the phenotype is already described in such a way that the operators can be applied directly to it. When this design is chosen, we speak of a direct representation. Mathematically this means that $f_g$ is the identity function, $f_g(x^g) = x^g$, the genotypic and the phenotypic space are identical (Rothlauf, 2005).

The decision whether to choose a direct or indirect representation and, if existent, how to design the genotype, is dependent on the problem. Often, when an indirect representation is chosen, the phenotype is mapped to standard genotypes. This is advantageous since they have been thoroughly investigated and standard operators exist. Such a classical representation are binary strings with fixed length as exemplified in figure 4.2. In the literature, GAs with such a representation are called canonical GAs. It is the most common representation for GAs focussing usually on recombination and using mutation only as background noise. This representation can be codified very easily and effectively and implemented with simple operators. In our application for the WDP and the MDKP, we could just interpret each bit in figure 4.2 to signify one decision variable. More precisely the figure could represent a solution where we accept bids 1, 2, 6, 7, and 8. Unfortunately, with this easy representation we do not take into account the constraints. The accepted bids might demand more units of goods that are available. Later on, we will see, how this problem can be solved.

A slightly more complicated representation might use strings of real-values, if it is used for encoding real-valued problems. Here, mutation often plays a more important role than recombination as it is hard to recombine real values in a meaningful manner. Yet, as we will see later, real-value representations are also helpful for permutation and binary combinatorial problems. In fact, the GA we focus on, uses such a real-value representation to solve the MDKP. For an overview of other representation techniques, like order-based and tree-based representation, the reader is referred to Rothlauf (2005).

4.1.2 Operators

The three genetic operators specify how the populations develop throughout the generations. The initial population is most often generated randomly. In our example with
binary strings, the alleles of an individual could be drawn randomly from a uniform distribution to be either 0 or 1. Afterwards the individuals are evaluated in order to determine which ones are allowed to create new offspring. Here, the first operator, the selection comes into play.

Selection

This operator selects the parents who will form the mating pool for the next generation. The size of the mating pool is dependent on the number of children to be created. In a classical generational GA a population of size $n$ generates $n$ children, while in most steady-state algorithms only one child is created per generation. A common selection procedure is tournament selection. In tournament selection, tournaments with size $k$, where $k$ is often two or three, take place between randomly drawn individuals. The individual with the highest fitness wins the tournament and is added to the mating pool. In some cases not all children will replace the former generation as it might be advantageous to keep some of the best individuals. We call this a replacement scheme with elitism (de Jong, 1975). Thereby, the selection pressure is augmented because it is even harder for a new offspring to become part of the new generation. In steady state algorithms often the two parents are chosen randomly without any selection procedure and therefore without selection pressure. That is why, the rules for the replacement of the old population have to control the selection pressure. Often, the new offspring only replaces individuals of the population if their fitness values are better. In other words the selection pressure is not controlled by choosing the mating pool but by choosing a strict replacement criteria. For an enumeration and explanation of different selection procedures, see Reeves and Rowe (2002, p.31).

Recombination

The recombination operator determines how parents create their children. It is called a sexual operator, since at least two individuals are needed for its application. In contrast, mutation as an asexual operator only works on one single individual.

Again, there exist some standard methods for recombination, being well explored. Just like representations have to be carefully chosen in accordance to the problem and its potential solutions, the recombination and the mutation have to be matched with the representation. The most common class of recombination operators works on string representations with a fixed length. The genotypes\(^1\) of the two parents are cut into several pieces and are rearranged. In this case, recombination is also denoted as crossover standing for the process of reordering subcomponents of the chromosomes (de Jong, 2006, p. 63). The simplest version of these recombination methods is the one-point crossover in which only one cutting point is defined. Here, the genotypes of the two parents, A and B, are respectively cut into two parts A1, A2 and B1, B2. The cutting position must be the same for both parents and it is normally determined randomly. When two children are created, child $\alpha$ is built by concatenating A1 and B2, and child $\beta$ by concatenating B1 and A2. When only creating one child, one just chooses one of the two concatenating options. More complex versions of this type of operator define more crossover points, at most $n - 1$.

\(^1\)If we have chosen a direct representation, the genotype is equal to the phenotype.
4.1 Functionality of Genetic Algorithms

where \( n \) is the length of the string. In the latter case, we speak of a uniform crossover. An example of the one-point crossover and the uniform crossover is illustrated in figure 4.3. With the amount of crossover points, one can influence the degree of similarity between offspring and parents. As a general rule, more crossover points mean more diversification in the search process as the children are less similar to their parents. That is because a lot of genes are reordered for the offspring’s creation.

Additionally to this classic \( k \)-point crossover for fixed length strings, there exist methods for non-linear representations, like permutations, or representations with variable string length. Some of them are discussed later in this chapter when introducing existing GAs for the MDKP. For an overview of different recombination operators the reader is referred to (Reeves and Rowe, 2002, p. 38).

**Mutation**

Mutation is the asexual reproduction operator working only on one parent. It basically copies the parent and modifies some genes to create the new offspring. In GAs mutation is often applied after the recombination process has already created the offspring. It then just changes some of the children’s genes. Yet, in other domains of evolutionary algorithms, like evolution strategies, mutation is the only reproduction operator. For controlling the intensity of modification induced by mutation, one has to decide on the number of genes to mutate and the strength of the modifications. For the second parameter we have to assume that some kind of distance metric is associated with a gene, otherwise a magnitude of change cannot be determined.

As an example, take the simple binary string. As each gene has cardinality two, the distance metric is naturally defined by the Hamming distance. If we flip one bit out of the string to its opposite, we have maximal distance of one to the original individuals and if not, the distance is zero as the gene is unchanged. Assuming that the string length is \( n \), the amount of variation depends, as well, on the number of genes we flip. Often, a size of \( 1/n \) is advised. For real-valued representations which have infinite cardinality per gene, one potential mutation operator is the Gaussian mutation with mean 0 and a self-defined deviation.
4.1 Functionality of Genetic Algorithms

4.1.3 Designing a Genetic Algorithm

In order to design a successful GA four aspects have to be considered (Mitchell, 1996, p. 156):

1. representation
2. selection
3. recombination operator and mutation operator
4. parameters, such as population size, recombination rate, and mutation rate

Those four decisions have to be made with regard to a successful search process. But where does the search exactly take place and what is meant by a successful search? As alluded to in the section about representations and fitness functions, the search takes place in a search space. Having chosen an indirect representation, there are two different search spaces: the genotypic one, denoted by \( \Phi_g \), and the phenotypic search space \( \Phi_p \). In the case of a direct representation, the GA directly searches in \( \Phi_p \). In the following paragraphs, the focus is on indirect representations as the analysis can be transferred easily to the direct case.

Each element in the genotypic search space maps to a phenotype, a solution of the problem. This mapping is surjective: each phenotype must be represented by at least one genotype. This guarantees that the genotypic search space contains all possible solutions, including the optimal one(s). Whenever the mapping is also injective, one genotype is exactly associated with one phenotype. In this case, we speak of a non-redundant representation: applying a random search in the search space, each solution would have the same probability of being chosen. Yet, sometimes, prior knowledge about high-quality solutions is given. If so, it can be of use to bias the search by over-representing phenotypes with high fitness, as the probability to find a solution of high-quality in the search space will increase.

The definition of a metric is crucial for the analysis of the search space. The distance \( d(x_a, x_b) \), based on this metric, determines the similarity between two individuals \( x_a \) and \( x_b \): the smaller the distance, the higher the similarity. The neighborhood of individual \( x_a \) includes all individuals with minimal distance to \( x_a \), where \( d_{\text{min}} \neq 0 \) (Rothlauf, 2005). Using, for instance, the Hamming distance for binary representations, \( d_{\text{min}} \) is 1. With an indirect representation it is necessary to define both a metric \( d_p \) on \( \Phi_p \) and \( d_g \) on \( \Phi_g \). While \( d_p \) is predetermined by the similarity of solutions for the given problem, \( d_g \) depends on the genotypes chosen. Consequently, the two metrics differ in most cases. We will see later in the concept of locality that for the success of a GA, it is fundamental that the neighborhood of a genotype maps to the neighborhood of the corresponding phenotype.

With the definition of a metric, the structure of the search space is determined. The number and positions of the optima and minima are hereby most relevant. Having a large number of local optima for a maximization problem, the search process can easily get stuck in one of those local optima and fail to find the global one. This behavior is, as well, dependent on the basins of attraction of the optima. Once the whole population has fallen into such a basin, it is hard to escape from there. Therefore the search space and
the operators must be designed so that the basins of attraction do not lead to premature convergence. On the other hand, a search space without any basin of attraction would make it hard to find optima at all. The mentioned aspects of search space and others are investigated in the field of fitness landscape-analysis (Reeves and Rowe, 2002, p. 231).

Coming back to the question of what a successful search is, we have to think about how to move through the search space in order to find a good solution. The GA should converge with all individuals being in the neighborhood of the best solution. If, however, the search process concentrates too early on a small subset of the search space, causing premature convergence, the algorithm has probably overseen better solutions in other regions. To cope with this trade-off between specification and diversification in the search process, is the key for a successful search. Holland (1975) speaks of a "tension between exploration and exploitation" for describing the trade-off between trying out new possibilities and browsing the search space (exploration), then again, incorporating and using past experience for an exhausting search in the region of good solutions (exploitation) (Mitchell, 1996). All heuristics which work by searching the solution space suffer from the problem of ensuring diversification of the population while focussing the search. Holland tried to show with his schema hypothesis that under certain assumptions the GA succeeds in balancing out this trade-off (Mitchell, 1996, p. 119). For an introduction to schema theory, the reader is referred to Holland (1975) and Reeves and Rowe (2002, p. 65).

In the literature the selection in a GA is most often identified with the intensification of the search process, while the recombination and the mutation are understood as the diversification or variation operators. Mitchell (1996) argues that a high selection pressure yields high-quality, but, causes simultaneously, suboptimal individuals to take over the population, leading to premature convergence. Diversification is prevented and further progress is hampered. Contrariwise, a low selection pressure results in unfocussed search and slow progress. In the worst case, convergence is made impossible.

Raidl and Gottlieb (2005) introduce a formal model to characterize the interplay of selection, mutation, and recombination when looking at the MDKP. They consider three aspects as being important for a good performance: locality, heritability, and heuristic bias. Locality is fulfilled if small steps in the genotypic search space cause small steps in the phenotypic search space. This notion is important for the exploitation of promising regions by the mutation operator. The hope is that close to good solutions other good solutions are situated. A metric in which the distances $d_p$ and $d_g$ are correlated in the sense of locality, prevent a search from being a random search. Heritability describes the ability of recombination to transmit valuable properties from the parents to the offspring. Yet, just copying features of the parents would lead to stagnation. The trade-off between exploitation of parents' properties and the incorporation of new properties must be solved by reproduction. Heuristic bias measures the influence of heuristics on the search process. As already alluded to, the GA must incorporate problem-specific knowledge. This can be achieved by biasing the mapping from genotypes on phenotypes with higher fitness. Here, the trade-off consists in the conflict between too much bias, leading to a decreased exploration and too little bias leading to a disregard of prior knowledge. Too little heuristic bias, potentially, results in unfocussed search or bad solutions.

Other possibilities to ensure a certain degree of diversity are duplicate elimination and niching (Gottlieb, 1999a). In parallel niching methods, sub-populations are formed and
maintained within the space of a single population, and in sequential niching methods, sub-
sub-populations are formed and maintained over time (Mahfoud, 1995). Duplicate elimination
has proven its success in steady-state GAs. The population is not allowed to have identical
individuals, where identical can be defined on genotypic or phenotypic basis. That effect
is reached by initializing the population at the beginning with only non-duplicates and by
rejecting all children being equal to an individual in the existing population. Duplicate
elimination has proven to be a very simple but effective method to ensure diversity in the

Having this knowledge about the trade-off between exploration and exploitation in mind,
the three operators must be designed. What remains is the choice of the fourth aspect:
the parameters. The parameters mainly comprehend the population size and the ratios
for reproduction. The population size is often chosen between 50 and several hundreds.
It reflects, once again, the trade-off between diversification and specification. A small
population size probably is less diverse than a large one. A large population might be too
unfocused. Additionally, to evolve a huge population consumes much more computational
resources. For recombination and mutation a rate is needed which determines in how
many cases a variation takes place. A common choice for the mutation rate of strings
is, for example, $1/n$, where $n$ is the length of the string. Each bit is then mutated with
probability $1/n$. For canonical GAs the recombination rate is normally much higher than
the one for mutation and can be up to 100%. The rate determines the probability with
which two parents out of the mating pool recombine to create their offspring.

One problem of parameter setting is the non-linear interaction among them; hence, they
cannot be optimized separately. In fact, it seems unlikely that any general guidelines can be
found a priori. To tackle this challenge, an attempt at applying a second GA for optimizing
the parameters before or during the search process was first made by Grefenstette (1986).
His results were slightly but significantly better than the ones achieved by an earlier study
with systematical changes in the parameter settings (de Jong, 1975). Trying to optimize
the parameter settings during the run is known as self-adaptation. Most believe that due
to the missing knowledge about the parameters a priori, adaptation during the search is
the most promising method. In some areas, like evolution strategies, the mutation rate
is often encoded directly in the string and changed throughout the search process (Davis,
1989). In our viewpoint, self-adaptation is a central theme for improving performance of
GAs. We think that the best approach is to gather as much as possible problem-specific
and test instance specific knowledge, to achieve a well-suited adaptation.

To sum up, in this section we have learned about the general functionality of a GA
and we know basic guidelines to focus the search. The main aspect we have focused
on is the trade-off between exploration and exploitation. In the following section, the
gained knowledge is applied to our problem domains. First, the problem-specific search
space is described, and then different methods for biasing the search in form of bid-sorting
heuristics are presented. Finally, with all these concepts in mind, an overview of existing
GAs for the MDKP is given. A deeper analysis of the specific weight-coded approach
follows in the next chapter.
4.2 Search Space of the WDP and the MDKP

As noted earlier, choosing a direct representation there only exists a phenotypic search space, $\Phi_p$, and for indirect representations additionally a genotypic search space, $\Phi_g$, subsists. Since the characteristics of $\Phi_g$ depends on the chosen representation, in this section, we mainly consider $\Phi_p$.

In the WDP the phenotype expresses which bids are accepted and which are refused, while in the MDKP it specifies which items are packed in the knapsack and which are left outside. So, in both cases, it tells if the decision variable $x_j$ is 0 or 1. To represent this vector of decision variables, there are basically two options. The first option is to list the decision variables set to one with their indices. In the WDP the phenotype 2, 5, 10 expresses that bids 2, 5, and 10 are taken. However, from a computational viewpoint, a much better choice is to take a bit string with length $n$ and to set each bit $j$ to 1, if bid $j$ is chosen. Another advantage of this representation is the easy distance metric it induces. The distance of two individuals, $\text{ind}^1$ and $\text{ind}^2$ is determined by the Hamming distance $d^h$. The Hamming distance is the number of bit positions $b_j$ were the two individuals are different:

$$d^h(\text{ind}^1, \text{ind}^2) = \sum_{j=1}^{n} |b^1_j - b^2_j|.$$ \hfill (4.1)

As neighbors we define all individuals with minimal distance of 1. Considering a problem with $n$ decision variables, the search space can be represented as an undirected graph with $n$ nodes and an edge between nodes if they are neighbors, exemplified in figure 4.4.

An important structure of the search space is the division in a feasible part and an infeasible part. In the feasible region all phenotypes can be found which fulfill the constraints and in the infeasible region all those violating constraints. As an example take a problem with three decision variables. The corresponding graph of the search space is shown in figure 4.4. Consider the constraints:

$$3x_1 + 2x_2 + x_3 \leq 4 \quad \text{(4.2)}$$
$$x_1 + 2x_2 + x_3 \leq 3. \quad \text{(4.3)}$$

In the figure the infeasible region is shaded grey. We now define the boundary $B$ of the search space as all feasible solutions to which we cannot add a 1 without creating an infeasible one. In the example $B = \{101\}, \{011\}$. For all feasible solutions which are not part of the boundary, their value can be improved by switching a 0 to a 1. Therefore, the optimal solution of the problem must be in the boundary. The awareness of this fact is fundamental for guiding the search. The algorithm should always try to find solutions in the boundary, as the optimum of the problem must be located there (Gottlieb, 1999a).

A further crucial aspect of the search space is the difficulty to find feasible solutions at all. In most cases a mapping from infeasible solutions to feasible solutions will be necessary, being it from the infeasible phenotypic into the feasible phenotypic region, or from the genotypic space to the feasible phenotypic region. As the feasible region is usually smaller than the whole search space, this mapping comes along with redundancy. Many infeasible solutions must be mapped to one and the same feasible one. A highly redundant mapping bears advantages and disadvantages which must be taken care of. One
4.3 Problem-Specific Bias by Bid-Sorting Heuristics

Most non-optimal algorithms for the MDKP and the MUCA use some version of the primal greedy heuristic to bias the search process to promising regions. To this end, they sort the bids according to some criteria with a bid-sorting heuristic. Sandholm (2006) notes that even for the exact branch and bound method it is useful to order the bids to enhance pruning for upper bounding. By first investigating the branches with the most promising bids, one can expect to obtain higher upper bounds. These improved bounds will fasten the search afterwards since branches can be pruned more easily. Furthermore, guiding the

---

Figure 4.4: Search space illustrated as graph. Nodes which are connected with edges represent neighbored solutions. The shaded nodes mark the infeasible search space.

advantage is the achievement of a bias towards promising regions by mapping primarily to high-quality solutions. This idea is dealt with in the next section. A disadvantage is the high inefficiency of the search since many individuals are mapped to the same solutions. This aspect plays a major role in the analysis of phenotypic duplicates in the next chapter.

Along with this mapping locality problems can occur. The mapping should be designed so that neighbored infeasible solutions are mapped to neighbored feasible ones. This undertaking, however, is quite difficult and complex. First, in most cases, many 1s in the infeasible solution must be changed to a 0 to represent a feasible solution. The question is, which decision variables to change. The high number of mapping possibilities makes it harder to find a local one among them. Second, the decision whether to set only one single decision variable to 1 or not can already make huge differences in the solution. Imagine two neighbored infeasible individuals, \( \text{ind}_1 \) and \( \text{ind}_2 \), \( \text{ind}_2 \) has one more 1 than \( \text{ind}_1 \). According to the notion of locality, the mapping should map both infeasible solutions to a neighborhood in the feasible region, since \( d^b(\text{ind}_1, \text{ind}_2) = 1 \). If the mapping includes the additional 1 in the feasible phenotype, more resources are consumed. Hence, decision variables which can be set to a 1 in the mapping of the very similar \( \text{ind}_1 \) must potentially be changed to 0. Contrariwise, if the mapping always ignores additional 1s in the neighborhood all infeasible solutions would be mapped to the same feasible solution\(^2\). To conclude, as the search space is discrete, it is very hard to determine a mapping which guards locality and in some cases it might even be impossible.

---

\(^2\)Any combination from \( \{0...0\} \) to \( \{1...1\} \) is possible by just adding iteratively a 1 to \( \{0...0\} \).
4.3 Problem-Specific Bias by Bid-Sorting Heuristics

search so as to finding good solutions earlier leads to a better anytime performance in case the algorithm is stopped before having reached the optimum.

As bid-sorting heuristics are the key components responsible for the success of the GA we analyze in this work, we now present some of the most important ones. The notion of bid in the domain of MUCA is equivalent to item in the domain of MDKP. That is why, from now on, these two words will be used interchangeably. Instead of bid-sorting heuristic, we could as well speak of item-sorting heuristic.

**Bid Price (BP)** The easiest way to sort the bids is by prices. Consider the bid with the highest price first and accept it, if no constraints are violated. Then, check the bid with the second highest price, etc. Obviously this sorting-heuristic is too simple, since it does not take into account the goods desired in the bid. Consequently, it is impossible to determine whether the bid is totally low-priced or high-priced.

**Normalized Bid Price (NBP)** Thinking about which bids to choose first, a more natural choice is to sort them according to the following ratio (Dobson, 1982):

\[ \frac{p_j}{\sum_{i=1}^{m} r_{ij}} \]  

(4.4)

In words, the price offered for a bid \( j \) is divided by the number of units demanded in the bid, so the average price per unit is computed. For instance, if an agent bids 10€ for 2 copies of \( \text{good}_1 \) and 3 copies of \( \text{good}_2 \), the ratio is \( 10/5 = 2 \). Apparently all goods are treated equally in this ratio irrespective of the scarcity of the capacities \( c_i \).

In the combinatorial auctions community a more general version of this ratio was proposed for the single-unit case \( (r_{ij} \in \{0, 1\}) \) (Lehmann et al., 1999):

\[ NBP_j := \frac{p_j}{(\sum_{i=1}^{m} r_{ij})^l}, \quad l \geq 0. \]

(4.5)

In the same work it was shown that for \( l = 0.5 \), the according primal greedy heuristic ensures the best worst-case bound one can hope for. It approximates the optimal allocation within a factor of \( \frac{1}{\sqrt{m}} \). In a later paper, the same bound was proved for the multi-unit case (Gonen and Lehmann, 2000). Due to the good results of the NBP, we will incorporate this bid-sorting heuristic with \( l = 0.5 \) in our analysis. Please note that, in case the NBP is applied to a branch and bound algorithm instead of the primal greedy heuristic, it has been tested empirically that \( l \in [0.8, 1] \) yields the best results for pruning (Sandholm et al., 2005).

**Relaxed Linear Program Solution (RLPS)** Another idea to sort the bids is to compute the relaxed linear program as described in section 3.1. The bids are just ordered decreasingly according to their relaxed LP solution. The LP relaxation assigns continuous values from 0 to 1 to the decision variables \( x_j \). Remember that \( x_j \) is 1 in the original problem if the bid is accepted and 0 otherwise. Correspondingly, for the relaxed LP this means that bids with a high value equal or close to 1 should be preferred to bids with a low value, close to 0. The justification behind that idea is that "the more of the bid is accepted in the relaxed LP, the more likely it is to be competitive" (Sandholm et al., 2001, p. 4).
Scaled Normalized Bid Price (SNBP) The drawback of the NBP that treats all capacities equally is taken care of in this heuristic. Fox and Scudder introduced a relevance value \( \mu_i \) which expresses the scarcity of the capacity,

\[
\frac{p_j}{\sum_{i=1}^{m} \mu_i r_{ij}}.
\]

(4.6)

Whenever the relevance value is high, the fraction decreases expressing that it is unfavorable to choose this bid. There exists one such value for each good \( i \) with the capacity \( c_i \). The idea is to choose a high value for a constraint which is rare, as to punish consumption of that resource. What remains is the choice of the relevance value. It is suggested to set \( \mu_i := 1/c_i \) or \( \mu_i := (\sum_{j=1}^{n} r_{ij} - c_i) \) (Senju and Toyoda, 1968, Kellerer et al., 2004). In the following we denote SNBP with:

\[
SNBP_j := \frac{p_j}{\sum_{i=1}^{m} r_{ij} c_i}.
\]

(4.7)

Shadow Surplus (SS) Pirkul (1987) proposed to take surrogate multipliers \( a_i \) as relevance value \( \mu_i \). Surrogate multipliers aggregate all capacity constraints to a single one by using weighted sums. This problem is called the surrogate relaxation problem of the MKP:

\[
\begin{align*}
\text{maximize} & \quad f = \sum_{j=1}^{n} p_j x_j \\
\text{subject to} & \quad \sum_{i=1}^{m} a_i \left( \sum_{j=1}^{n} r_{ij} x_j \right) \leq \sum_{i=1}^{m} a_i c_i \\
\text{with} & \quad x_j \in \{0, 1\}, \quad j = 1, \ldots, n \\
& \quad p_j > 0, \quad r_{ij} \in \mathbb{N}.
\end{align*}
\]

(4.8)

(4.9)

He suggests several possibilities to derive surrogate multipliers. One simple and useful method is to use the dual variables of the relaxed LP (Pirkul, 1987). They serve as a proxy of how valuable a unit of a good is and are therefore called shadow price of the \( i \)-th constraint. For more information about relaxed LPs and their duals as applied for the MDKP, see Kellerer et al. (2004).

We can now define the shadow surplus (SS) of bid \( j \) to be:

\[
SS_j := \frac{p_j}{\sum_{i=1}^{m} a_i r_{ij}},
\]

(4.10)

where \( a_i, \ i = 1, \ldots, m, \) denote the solutions of the dual LP. The SS is sometimes also denoted as profit/pseudo-resource consumption ratio.

There are other bid-sorting heuristics which are specialized on branch and bound methods and are not useful for the primal greedy heuristic, like bid graph neighbors, number of items, and most fractional bid (Sandholm, 2006). They focus, for instance, on choosing the bids which have a value closest to 0.5 in the relaxed LP. The idea behind this it is that one should branch first on bids where the LP is more uncertain.
4.4 Existing Genetic Algorithms for the MDKP

Among the different heuristic optimization methods for solving the MDKP, GAs have proven to be very efficient. Surprisingly, in the field of combinatorial auctions, evolutionary methods are more or less unstudied, as demonstrated in the literature review of chapter 3. Therefore, in this section, only GAs designed for the MDKP are surveyed.

The core of designing a successful GA for the MDKP, and for any other constraint optimization problem, is to find an appropriate constraint handling technique. Given that many possible occurrences for the set of decision variables are impossible, we carefully have to think about how to represent solutions. Furthermore, the analysis of the phenotypic search space pointed out that the most promising search techniques concentrate on the boundary of the phenotypic search space. On the basis of those two aspects, a natural classification of constraint handling heuristic optimization methods is done by two criteria: representation and region of phenotypic search space explored. Some algorithms focus on the complete search space allowing individuals to represent infeasible solutions. Others concentrate from the beginning only on the feasible search space. This observation yields to the following classification (Gottlieb, 1999a):

1. direct representation with search in the complete search space,
2. direct representation with search in the feasible search space, and
3. indirect representation with search in the feasible search space.

To introduce a class for indirect representations in the complete search space is unnecessary. It would mean mapping the genotype to an infeasible phenotype which afterwards is mapped to a feasible phenotype. By integrating both mappings into one, one can end up with exactly the third class.

4.4.1 Direct Representation

As discussed in section 4.2, a binary string is chosen as direct representation. The length of the string corresponds to the number of decision variables. A sequence of the decision variables must be pre-defined so that each position in the string stands for exactly one such variable. In the case of MDKP, a one on string position $i$ signifies that item $i$ is chosen to be packed in the knapsack. In this section we will give an overview of such direct binary string representations for the two search options: complete vs. feasible search space.

Complete Search Space

In this approach, infeasible solutions are allowed to be part of the population. Therefore, a mean to guide the search to feasible high-quality solutions is to adopt the fitness function by punishing all infeasible solutions. Naturally, the fitness function for the MDKP is chosen to be the objective function $f(x)$ of the IP formulation, defined on each individual $x$. By introducing a penalty function, $\text{penalty}(x)$, $f(x)$ is replaced by a new fitness function $f_{\text{new}}(x)$ (Gottlieb, 1999a):
The penalty function determines the relationship between feasible and infeasible solutions. It must be chosen carefully in the light of the problem. If the penalty is only moderate, the search might lose focus on the feasible solutions. Otherwise, if it is chosen too strictly, feasible regions might stay unexplored since it might be necessary to cross infeasible regions to reach them. A good choice for guiding the search in promising regions is a penalty function that estimates the distance to the feasible search space. Thereby, infeasible regions lying close to the boundary are favored to those outlying infeasible regions, where too many changes in the individual have to be made to fulfill the constraints.

According to Gottlieb (1999a), a major problem are new local optima which are potentially generated by $f_{\text{new}}$. That is why he suggests ensuring that the local optima with respect to the search space and $f_{\text{new}}$ are equal to the ones with respect to $f$ and its search space. The discussed aspects lead to two advices for designing a penalty function:

1. The new fitness landscape induced by $f_{\text{new}}$ should not contain local optima corresponding to infeasible solutions.

2. The new fitness landscape should measure the distinction between infeasible and feasible search space by means of the distance to the feasible search space.

In literature, the most common problem with penalty functions is the feasibility problem (Gottlieb, 1999a). It describes the inability to guide the search towards feasible solutions. The main reason that this problem occurs seems to be based in the initialization phase. Gottlieb examined that once only infeasible solutions are generated at the beginning, the tested algorithms fail to find any feasible solutions. Overall, most algorithms using penalty functions seem very sensitive to the initial population. Furthermore, much computational resources might be wasted by an unguided search while exploring only the infeasible region. This holds, specifically, for penalty functions introduced by Hoff et al. (1996), who base the penalty on the sum of all constrained violations and Khuri et al. (1994), who measure the distance by the number of violated constraints.

In his PhD thesis Gottlieb (1999a) tried to focus the search by introducing several monotony concepts for the new fitness function. The main idea behind these concepts is that each increase of the penalty term must dominate the corresponding increase in $f$. The penalty function which implements the most strict monotony concepts provides good results, but still worse than the ones of Raidl’s (1999) algorithm, we focus on in this work.

As the standard binary string representation is used for direct search in the whole search space, standard operators like k-point crossover and bit-flipping mutation can be applied. In the direct representation no mapping occurs. That is why a discussion about redundancy or mappings ensuring locality is needless.

Generally, a reason which might hinder this approach to be the best one is missing locality in relation to the fitness function. Neighbored solutions might have quite different
4.4 Existing Genetic Algorithms for the MDKP

fitness values, especially if one is feasible and one infeasible. Gottlieb deals with this problem with the help of the mentioned monotony concepts. Moreover, in our view, the bias towards good solutions should be more intense. With his concept, Gottlieb only biases the search towards the boundary of the search space, but not towards good solutions in the boundary. Together with inefficiency inherent in representations in which also infeasible solutions are encoded, this missing heuristic bias might be the cause, why penalty functions have not been the most successful method to deal with constraints.

Figure 4.5: Illustration of a direct representation with repair mechanism and local optimization. All infeasible solutions are first mapped to the feasible search space. Afterwards the local optimization method maps the solutions into the boundary of the feasible search space.

Feasible Search Space

Direct search in the feasible region is potentiated by applying operators which guarantee the feasibility of the offspring. In most cases such operators get very complicated, which is why one splits the operators into two parts: (1) variation by standard operators and (2) repairing by mapping an infeasible solution to the feasible search space. Often a third phase for biasing the individuals to promising regions of the feasible search space is added in the form of a local optimization method. This mapping procedure is illustrated in figure 4.5. We think that the third phase is the one which is the missing part in the previously discussed search with penalty functions.

Repair methods have to guarantee certain properties in order to work successful. Among them are a polynomial running time and locality. Basically, these are the same requirements needed for mutation and recombination as the repair mechanism is just a part of those operators. Additionally to local optimization, repair operators can bias the mapping.

One effective repair technique for the MDKP was proposed by Chu and Beasley (1998). They first apply a uniform crossover and a bit-flip mutation to produce offspring. Afterwards, a repair phase DROP and a local optimization phase ADD are executed. In the DROP phase, bids are removed systematically from the current solution by applying a bid-sorting heuristic. All items set to one in the current individual are sorted in increasing order by the SS sorting-heuristic and removed one by one until the solution becomes feasible. In case a solution is already feasible at the beginning of the repair phase, the repairing has no effect on the solution. As the newly generated individual does not necessary lie in the boundary of the search space, the ADD phase finalizes the process. It sorts all
not-taken items of the individual, this time in decreasing order so that the most valuable item not already set to one is first. Finally, the whole list is traversed, adding items as long as they do not violate a constraint, guaranteeing to reach a solution in the boundary of the feasible search space. Chu and Beasley’s algorithm leads to very good results and is, hence, one of the state-of-the art algorithms. Later, it was improved by biased initialization methods which use a RLP primal greedy heuristic (Raidl, 1998, Gottlieb, 1999a).

Raidl and Gottlieb (2005) argue that strong locality and heritability together with the high heuristic bias are reasons for the success of the algorithm by Chu and Beasley. They apply different analyzing methods which support their observations. The necessary diversification in the population is reached by duplicate elimination.

In our opinion, it would be more accurate to just mention the strong heuristic bias as reason for the high performance. In fact, the DROP and the ADD phase lead to an extremely biased representation with a very high redundancy. Whole neighbored regions in the search space are mapped to the same feasible phenotype. In other words, the high locality is a result of the high heuristic bias.

### 4.4.2 Indirect Representation

As pointed out before, in indirect representations genotypes are mapped to the phenotypic feasible search space. In the genotypic space constraints are ignored. The mapping from genotype to phenotype is often denoted as decoder since it decodes a genotype to a feasible phenotype. The most promising approaches are the ones which map into the region where the optima are located; the boundary of the search space. This process is demonstrated in figure 4.6.

It was discussed in the previous section that mappings in the problem domains we consider suffer the locality problem and a high redundancy of the search space. With these aspects in mind, the most important indirect representations are now presented.

![Figure 4.6: Illustration of an indirect representation with a mapping from the genotypic space to the phenotypic space. The decoder is biased so that all genotypes are mapped to phenotypes in the boundary of the search space.](image)

---

3Note that this approach is very similar to the dual greedy heuristic introduced in 3.2.2 with the only difference that before the DROP phase not all variables are set to 1.
4.4 Existing Genetic Algorithms for the MDKP

Ordinal Representation

Using an ordinal representation, an ordered list $L$ of all items has to be constructed. The list assigns an index to each bid while the sorting is not of any importance. The genotype is a bit string $v = (v_1, ..., v_n)$ where each position $v_k \in \{1, ..., n - k + 1\}; \; k = 1, ..., n$. Then, a permutation of the items is generated by taking the first position $v_1$ out of the list, then the second and so on. Note that the upper value for $v_k$ diminishes by one for each position corresponding to the shrinking of the length of the list. Finally, the permutation is traversed in order and each item is checked for inclusion.

This representation has had only limited success compared with state-of-the-art methods like the repairing algorithm by Chu und Beasley (1997). We see the reason for this in locality and redundancy problems the representation suffers. Any reasonable definition of neighborhood in the genotype seems to evoke problems with the locality concept. This is because switching two positions in $v$, cannot only cause a switching of two positions in the final permutation, but can also cause a total reordering of the permutation. Imagine, a $v^1 = 5, 3, ...$ where the first two positions are switched to $v^2 = 3, 5, ...$ and a list $L = bid_1, bid_2, ...$. The resulting permutation of $v_1$ would be $bid_5, bid_3, ...$ while in the case of $v_2$ not only the first two positions of the permutation would be switched to $bid_3, bid_5, ...$, but to $bid_3, bid_6, ...$ This is because after removing the first bid of the list, the size shrinks by one and therefore, the fifth position of $L$ is assigned to $bid_6$. Thus, apparent small changes in the genotype can lead to huge changes in the phenotype.

In later representations we will see that the locality and the redundancy problem are often counteracted by a strong bias in the search. Specifically, the advantages of the redundancy can be utilized to lead to a very focused search. The only bias which is made on the ordinal representation is towards the boundary. This, apparently, does not suffice to guide the search through promising regions.

Permutation Representation

A permutation orders all items in some sequence $\pi = (\pi_1, ..., \pi_m)$. A decoder traverses this permutation one by one and adds the current items if no constraints are violated. In this way, solutions at the boundary of the search space are ensured. For reproduction of permutations, specialized operators are needed, like swap mutation or partially matched crossover (Goldberg and Robert, 1985). They have been investigated in particular for the travelling salesman problem, but also for the MDKP (Gottlieb, 1999a). The permutation representation has been applied several times for solving the uni-dimensional as well as the MDKP (Raidl, 1998, Hinterding, 1994, Thiel and Voss, 1994, Gottlieb, 2000). The results were acceptable, but not comparable with the best achieved by GAs.

The permutation representation has similar characteristics as the ordinal representation. One aspect which is improved is the higher locality by the mapping. Totally reordering problems, like the one described in the small example before, cannot occur. This is, probably the cause why it performs better than the ordinal representation.
Random-Key Representation

In this representation a vector \((w_1, ..., w_n)\) of weights \(w_j \in [0, 1]\) is used as genotype. Each position \(j\) represents an item. The decoder sorts all weights either increasingly or decreasingly. This yields, automatically, to a permutation of the items. In fact, the encoding is related to the permutation with the only difference that no special operators are needed (Hinterding, 1994, Hinterding, 1999). Standard operators like two-point crossover and Gaussian mutation are, for example, applied by Hinterding (1999).

The performance of this representation is worse than permutation because of missing locality. Furthermore, because of the real-valued genotype leading to a very large genotypic search space, the redundancy of the representation is even higher than in all other considered cases. Without heuristic bias advantages of a redundancy cannot be accomplished. Wherefore, the disadvantage of redundancy, mainly the inefficiency in the search process, dominates the search.

Weight-Coded Representation

Similarly as the random-key approach, the weight-coded representation uses weights as alleles. What differs is the decoding procedure. It consists of two steps. First, the prices for items in the objective function are biased with the weights encoded in the genotype. Thereby, the problem \(P\) itself is modified into a problem \(P'\). Second, a primal greedy heuristic including a bid sorting-heuristic is applied on \(P'\). The weight-coded approach has been successfully applied to different problems. For an overview, see Julstrom (1997).

Under a different name and with focus on search spaces instead of representations, the method got famous in sequencing problems (Storer et al., 1992, Storer et al., 1995). In this field, it is called problem space search. The idea in this domain is formulated as follows. The problem is perturbed with a distance smaller than a specified \(\epsilon\) which determines the neighborhood of the problem. A heuristic then searches for the best solution in that neighborhood. The objective function of the original problem is finally used to evaluate the solution to the modified problem. A similar idea is the heuristic search space where, instead of modifying the problem, the parameters of the applied heuristic are modified.

For the MDKP, Raidl (1999) used weight-codings with very good results, comparable with those of Chu and Beasley (1997). In fact, Raidl’s approach yields the best results among the GAs with indirect representation. That is why, we concentrate on this algorithm and point out important aspects in the next chapter. In our later analysis, we focus further on Raidl’s idea as it is a promising approach where improvements might still be possible.

For designing a weight-coded approach three parameters have to be chosen: the value of weights, the kind of modification by these weights, and the bid sorting-heuristic. Raidl suggested four methods to determine the weights and tested them with different strengths of perturbations \(\gamma\). As modifications, he proposed addition and multiplication. Concerning the bid sorting-heuristic, he examined two approaches: the already discussed SS and one using Lagrangian relaxation. The four suggested weights are:

**UNI**: Addition of uniformly distributed weights

\[
p_j' = p_j + w_j, \quad w_j = \mathcal{R}(0, \gamma p)
\]
Weights are drawn from a uniform distribution, \( R \). The range of \( R \) is from 0 to the average over original prices multiplied with the strength of perturbation \( \gamma \). The average is \( \bar{p} = \frac{1}{n} \sum_{j=1}^{n} p_j \).

**RELUNI: Addition of relative, uniformly distributed weights**

\[ p_j' = p_j + w_j, \quad w_j = R(0, \gamma p_j) \tag{4.14} \]

Weights are proportional to the price which is perturbed. Raidl calls this method *symmetrical* since the problem structure stays unchanged if prices are multiplied with the same constant values.

**LOG: Multiplication with logarithmically distributed weights**

\[ p_j' = p_j w_j, \quad w_j = (1 + \gamma)^{R(-1,1)} \tag{4.15} \]

Here, the weights are logarithmically distributed in the range \([1/(1 + \gamma), (1 + \gamma)]\). Since the median of this distribution is 1, median perturbed prices correspond to original prices.

**LOGNORM: Multiplication with log-normally distributed weights**

\[ p_j' = p_j w_j, \quad w_j = (1 + \gamma)^{N(0,1)} \tag{4.16} \]

Instead of a uniform distribution, a normal distribution with mean 0 and standard deviation 1 is chosen. Therefore, smaller changes are made with higher probability than larger changes. Very large modifications are possible, but improbable.

One of the four methods is chosen in advance and is kept for the whole algorithm. In the initialization phase all weight vectors are generated according to the respective function for \( w_j \). Furthermore, the mutation biases the weights with the chosen method with a mutation rate of \( 3/n \) per weight. Concerning the recombination, no perturbation of the problem takes place, a simple uniform crossover is done with probability 1.

For a better understanding of the algorithm, a deeper investigation of the four perturbation methods is necessary. All methods show a high dependence on \( \gamma \). However, a \( \gamma \) of 0.05, for instance, might cause a moderate change of prices in the LOGNORM case and almost no change in the UNI case. Hence, it is crucial to choose different values of \( \gamma \) in all four cases. By comparing the first two options, the UNI modification will provide a lot of modification assumed that the spread of prices is not too high. Although the RELUNI, on first glance, does not differ much from the UNI perturbation, in fact, it has a much stronger bias towards the original problem. The original price \( p_j \) plays a much more decisive role for determining \( w_j \) as the average treatment of prices in the first case. That means, an already low price has less chance to be updated with a high weight.

Since the third and the fourth method are mathematically more complicated as the first two variations, we plotted them in figure 4.7. The figure gives a first impression of the drastically increase of the weight when \( \gamma \) and the random variable grow above 1. Such values are only possible for the LOGNORM method, since in the normal distribution with probability 15.9% a value above 1 is drawn and with 0.28% above 2.
4.4 Existing Genetic Algorithms for the MDKP

In figure 4.7 a similar plot is shown. This time, different curves for fixed $\gamma$ are drawn. This perspective demonstrates in more detail the high sensitivity of the LOG and the LOGNORM distribution. In the LOG distribution the most extreme value which can be reached with a $\gamma$ of 2 is a weight of 3. In the LOGNORM distribution the extreme values are above 8.

It seems unreasonable to choose parameters which can lead to such high weights, since the modification of the original problem gets extreme. Nevertheless, Raidl tested different $\gamma$ up to 100 and showed that as long as $\gamma$ is larger than a minimal bound, the solution quality stays quite robust. Nevertheless, with increasing $\gamma$ it converges much slower. We think that tests with such huge perturbation sizes are not necessary. Specifically for the LOGNORM perturbation method $\gamma$s below 1 should be considered as they generate already very different weights.

To sum up, methods three and four are very sensitive to the random variable and the perturbation size and extreme values are, particularly, probable for the fourth versions. On the one hand, the bias to the original problem is strong because of the multiplicative composition between prices and weights, on the other hand, since extreme weights are possible, strong modifications can happen.

In his experiments, Raidl concluded that differences in performance between those four methods are small. The best solutions averaged over 10 test instance with 5 constraints, 250 items and a medium resource capacity can be seen in table 4.1 (taken from Raidl (1999, p. 5)).

The table validates our proposition that the perturbation sizes must be chosen differently for the various methods. The higher perturbation size in the LOGNORM case than in the LOG case, is caused by this special data set. Overall, a $\gamma$ of 0.05 was found to work best.
with LOGNORM. The low value of $\gamma$, as well, supports our idea that LOGNORM is the most sensitive method. As explanation for the slightly worse results of UNI, Raidl notes the asymmetrical distortion since the median of the modified problem does not correspond to the median of the original problem. Experiments with other input sizes strengthened those observations. Overall LOGNORM performed best with a perturbation size of 0.05. That is why, in the following chapter, we will concentrate on the LOGNORM method.

<table>
<thead>
<tr>
<th>weights</th>
<th>$\gamma$</th>
<th>gapR(%)</th>
<th>eval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNI</td>
<td>0.15</td>
<td>0.286</td>
<td>30930</td>
</tr>
<tr>
<td>RELUNI</td>
<td>0.10</td>
<td>0.273</td>
<td>32420</td>
</tr>
<tr>
<td>LOG</td>
<td>0.05</td>
<td>0.267</td>
<td>24430</td>
</tr>
<tr>
<td>NORMLOG</td>
<td>0.10</td>
<td>0.275</td>
<td>82340</td>
</tr>
</tbody>
</table>

We have discussed the value of the weights and the problem modification by these weights. Now, the third aspect, the mapping from the modified problem to a feasible solution is addressed. Raidl suggests two decoding procedures which bias the search to potentially good solutions in the boundary of the search space. The first is the surrogate relaxation based heuristic which is equivalent to the SS heuristic, and the second is a Lagrangian relaxation based heuristic. In Raidls’ experiments, the SS heuristic performed always better than the Lagrangian one, that is why, in our investigation, we will focus on $SS_j := \frac{p_j}{\sum_{i=1}^m a_ir_{ij}}$.

For an efficient usage of this heuristic, the pseudo-resource consumptions $\mu = \sum_{i=1}^m a_ir_{ij}$ with the surrogate multipliers $a_i$ can be determined, once, at the beginning of the algorithm, since this value stays unchanged. That is why for the decoding of a genotype, in each decoding step only the modified price $p_j' = w_jp_j$ is computed and then divided by $\mu$. Then, the items are accepted in decreasing order of these fractions.

To sum up, the idea behind this procedure is to apply a primal greedy heuristic to a modified version of the problem. The problem is modified by perturbing the prices in the objective function with weights. The primal greedy heuristic determines a solution for the modified problem by setting all decision variables to either 0 or 1. The fitness of this solution, however, is determined by computing the value of the original objective function.

The weight-coded algorithm features a strong redundancy comparable to the one by the random-key representation. Nevertheless, in contrast to that approach, Raidl’s method takes advantage of the redundancy in incorporating a strong heuristic bias. Thereby, promising regions in the boundary of the search space are over-represented. Yet, we think that the extreme redundancy of the representation is not dealt with in an adequate manner. The redundancy of real-valued alleles is so high that more attention should be addressed to deal with inefficiency in the search and a sufficient diversification. That is why, in the following chapter we will analyze several aspects of Raidl’s approach in detail. Thereby, we will concentrate on the questions of inefficiency and on the trade-off between exploration and exploitation. In more details, we will address the question to what extent the exploited regions depend mainly on the bid-sorting heuristic applied in the algorithm.
Second, we will test the degree of diversification in the population and the convergence behavior by measurement of distances. Furthermore, we will analyze the effect of $\gamma$ and the mutation rate on the number of produced duplicates. The focus will be to observe how the parameters can be adapted to structural properties of the test instances.

![Figure 4.8: Sensibility of weights to random variable and $\gamma$ for the function $(1 + \gamma)\times$. In the LOGNORM distribution the weight is very sensitive to high perturbation sizes $\gamma$, due to possibly extreme values for $x$.](image)

### 4.4.3 Comparison

To conclude this section, we present the performance of the best GAs applied to the MDKP. To our knowledge these are the best results which can be found in the literature. All instances are tested on state-of-the-art test sets for MDKPs: 270 test instances from Chu’s complete benchmark suite. This data set will be discussed in detail at the beginning of the following chapter.

The direct representation with repair and local optimization methods perform best, but are closely followed by the weight-coded approach. Unfortunately the number of evaluations until the best solution is found (evals) is not given in the paper by Chu and Beasley. In Gottlieb (1999, p. 219), however, this algorithm is re-implemented with an improved initialization and a gapR of 0.547, which is only slightly worse than this of the original paper. Gottlieb reports an average number of 261143 evaluations.

To sum up, the weight-coded approach seems the most promising one, since it reaches a gapR, comparable to the best one, but takes much less evaluations to reach this gapR and is therefore thought to be faster. Furthermore, in our opinion, the algorithm by Chu and Beasley seems to be already improved to its best, while in the weight-coding
4.4 Existing Genetic Algorithms for the MDKP

Table 4.2: Performance of best genetic algorithms for the MDKP.

<table>
<thead>
<tr>
<th>representation</th>
<th>gapR[%]</th>
<th>evals</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>penalty</td>
<td>1.094</td>
<td>748866</td>
<td>Gottlieb (1999)</td>
</tr>
<tr>
<td>repair and local opt.</td>
<td>0.54</td>
<td>(261143)</td>
<td>Chu and Beasley (1998)</td>
</tr>
<tr>
<td>weight-coding</td>
<td>0.589</td>
<td>51179</td>
<td>Raidl (1999)</td>
</tr>
</tbody>
</table>

many parameters might be tuned such as the heuristic chosen, the perturbation size, the mutation rate, and the method of modification. Moreover, a thorough examination of bid-sorting heuristics can as well help to understand Chu and Beasley’s algorithm better, since they apply the same SS heuristic as Raidl. Additionally, we assume that running the weight-coded approach with fixed parameters produces an overhead on duplicates, so that a self-adaptation of those parameters controlling exploration and exploitation dynamically will lead to better results or at least to faster convergence.
5 Analysis

In this chapter, existing approaches for the multi-unit WDP and the MDKP are analyzed. It is the main part of this work. Yet, before starting, we would like to recapitulate what has been accomplished so far. After the introduction to the multi-unit WDP and the equivalent MDKP, an overview about different solutions to both problems was given. Since the problems are NP-complete, the focus lay on heuristic optimization methods, in particular GAs. In the preceding sections we discussed different GAs for the MDKP, because they have been proven to be a very successful approach. Due to their success, we suggest to apply the latter, as well, to the multi-unit WDP. The aim of this chapter is to analyze several approaches. The focus is on structural properties of the test instances influencing the performance of algorithms. With this knowledge, we will be able to give advices on designing meta-heuristics, which will potentially lead to an improvement of existing approaches.

The chapter is subdivided into four parts. First, the high dependency between structural properties of test instances and the performance of algorithms will be discussed. Important facets which might affect the performance will be introduced. They will be explained on the basis of the test instances we have chosen. Second, the performance of four approaches will be tested. While doing so, we will keep in mind the structural properties of the test data explained in the first analysis. The third and the fourth part will concentrate on Raidl’s weight-coded approach. In both parts, aspects of the search behavior will be analyzed. First we will address the diversification measured by distances. We will examine the heuristic bias which potentially hinders the population from staying diverse, as well as the convergence behavior of the algorithm. Furthermore, the contribution phenotypic duplicate elimination yields to diversity will be inspected. The last part will deal with the number of produced duplicates. The affect of parameters on this number is examined. Thereby the question will be answered, whether the algorithm can be improved by diminishing the amount of duplicates. All analyses will show whether a more specialized or a more diversified approach is adequate to solve the problems. In other words, we will answer the question how the trade-off between exploitation and exploration should be dealt with.

5.1 Test Instances

At the latest with the No-Free-Lunch theorem, it is common knowledge that no general algorithm can work well on all optimization problems. Researchers have addressed this problem by designing problem-specific algorithms which perform well on a selected set of problems. When incorporating problem-specific knowledge in the design of an algorithm, often the properties of test instances play a major role. Therefore, for making experiments with existing algorithms designed for the MDKP and MUCA, it is fundamental to analyze the test instances first. With the so gained understanding, it should then be possible, to find an adaptation of the algorithms to the underlying problem structures.
Research on CAs has profoundly addressed the subject of problem instances. There are mainly two reasons, why it is of particular interest for this problem domain. First, research in the WDP supports the already mentioned NFL theorem. A strong influence of bid distributions on computational performance has been found (Andersson et al., 2000). Second, overall there exist only few real-world data sets. One source is the data saved from the FCC spectrum auctions, but this source is still very limited. Therefore, the only solution is to generate artificial test instances.

Some researchers have addressed this topic by analyzing the structure of test data trying to generate hard test instances (de Vries and Vohra, 2003, Andersson et al., 2000). Furthermore, an attempt to generate realistic data was done by Leyton-Brown et al. (2000a) in the so-called CA test suite "CATS" (Leyton-Brown and Shoham, 2006). This test suite generates CA instances for the testing of single-unit WDP algorithms. In the five distributions offered, the authors try to reflect realistic, economically motivated domains. Since the first version of CATS in 2000, a second version has been developed. This version incorporates knowledge from work published by Leyton-Brown et al. (2003) who generate harder instances of the first CATS version (Leyton-Brown et al., 2006). Their approach is to use machine learning techniques to predict running times and the success rate of algorithms. These techniques use feature extraction to examine what structural components of test instances increase the WDP difficulty. They test the generated instances with four algorithms, CPLEX, the one from Gonen and Lehmann (2000), a simple branch and bound, and CASS (Fujishima et al., 1999). As all algorithms work with different mechanisms, a certain degree of generality of the results can be assumed.

Different features of single-unit test instances are indicated to have an influence on the hardness. For their understanding, we first have to clarify the notions of bid graph (BG) and bid-good graph (BGG). In a BG one node corresponds to one bid and nodes are connected with an edge, if the bids cannot appear together in the same allocation. A BGG is a bipartite graph with a node for each bid and a node for each good. An edge is between a bid and a good if the good is included in the bid. Some of the structural properties are:

- BG edge density,
- the clustering coefficient in the BG graph, reflecting the number of edges among neighbors, and
- BGG average and minimal good degree of the good nodes.

In the MDKP literature, the structure of problem instances is controlled by the tightness ratio $\alpha$. This ratio expresses the scarcity of capacities, defined as:

$$\alpha_i = \sum_{j=1}^{n} \frac{c_j}{r_{ij}}$$  \hspace{1cm} (5.1)

A tightness ratio of 0.75, for instance, expresses that 75% of the desired units are available. The lower the tightness ratio, the more restricted is the MDKP.

1See http://cats.stanford.edu/.
5.1 Test Instances

The question remains whether a highly or weakly restricted problem causes harder test instances. For search algorithms which explore the complete search space, probably highly restricted problems are harder as more unfeasible solutions exist (feasibility problem). By contrast, for methods concentrating on only the feasible region, weakly restricted problems should be harder as the feasible region is larger than in highly restricted problem instances.

To sum up, we now have some criteria to analyze the hardness of test instances. The edge density and the clustering coefficient, just like the tightness ratio can be applied directly to multi-unit problems and reflect the degree of restriction. With the BGG good degree, interpretations must be done with caution as the concept cannot be translated directly. A good node having two edges in the BGG graph in the single-unit case expresses that only one of the two connected bids can be accepted, while in the multi-unit case, no such conclusion can be drawn. Nevertheless, the BGG might be very important, since it is determined by the number of goods included in the bids. This number reflects the number of non-zero entries in the constraint matrix, denoted as density. We think that a dense matrix makes an instance more difficult, because it is harder to divide the problem into sub-problems. Furthermore, all goods must be considered in all bids, increasing the complexity and the cost of computation.

To prevent confusion, we would like to clarify the difference between density of the constrained matrix and degree of restriction. The density expresses the amount of non-zero entries in the matrix. Whenever bids include only a small number of the available goods, the density is low. Yet, the density does not express whether the instance is highly or weakly restricted. The extent of restriction depends mainly on the number of units included in a bid. Only if a lot of the available capacity is desired in the bids, they exclude each other and lead to a high restricted problem. A sparse matrix can, for example, be highly restricted if the bids always desire the maximal number of units available per good.

We have decided to carry out the experiments with three problem types, two from the MUCA literature and one from the MDKP literature. The choice for the test instances from MDKPs was quite clear while it was hard for the other case. Unfortunately, for MU-CAs there are even fewer test instances available than for the single-unit case. Data from real-world applications is not available at all. Actually, in literature we have found only one method for generation of MUCA instances (Leyton-Brown et al., 2000). It was used by both algorithms addressing this subject (Gonen and Lehmann, 2000, Leyton-Brown et al., 2000). We have chosen this method in addition to one out of the PATH test suite from CATS and adopted it to MUCAs. The two problem types are denoted with LB and MULTIPATH. In MDKPs, typically, test instances are taken from the OR-library\(^2\). From this library we have chosen the benchmark suite which was used for experiments with the successful GA by Chu and Beasley (1998), described before. These instances are from now on referred to as MKNAPCB. The LB data generation from Leyton-Brown et al. (2000b) works as follows:

1. Set the number of available units for each good:
   a) For each good \(i\), randomly choose \(units_i\) from the range \([1, units_{max}]\).
   b) If \(\sum_i units_i \neq \frac{m \cdot \sum_{j=1}^{units_{max}} units_{max}}{units_{max}}\), the expectation on \(\sum_i units_i\), then go to 1.a).
   This ensures that each trial involves the same number of units.

\(^2\)See http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html.
2. Set an average price for each good: \( \text{avgprice}_i \) is drawn uniformly randomly from the range \([\text{avgprice}_{\text{base}} - \text{avgprice}_{\text{var}}, \text{avgprice}_{\text{base}} + \text{avgprice}_{\text{var}}]\).

3. Select the number of goods in the bid with a decay distribution:
   a) Randomly draw one good without replacement,
   b) with probability \( \text{prob}_1 \), if more goods remain then go to 3.a).

4. Select the number of units of each good, according to another decay distribution:
   a) Add a unit,
   b) with probability \( \text{prob}_2 \), if more units remain then go to 4.a).

5. Set a price for this bid: \( p_j = R[1 - \text{price}_{\text{var}}, 1 + \text{price}_{\text{var}}] \times \sum_{i=1}^{m} (\text{avgprice}_i \times r_{ij}) \), where \( R \) is a random uniformly drawn number.

The parameters in the paper are fixed as follows:

\[
\begin{align*}
m &= \{10, 14\} \quad \text{(number of goods)} \\
n &= \{1500, 2500\} \quad \text{(number of bids)} \\
units_{\text{max}} &= 5 \\
\text{avgprice}_{\text{base}} &= 50 \\
\text{avgprice}_{\text{var}} &= 25 \\
\text{prob}_1 &= 0.8 \\
\text{prob}_2 &= 0.65 \\
\text{price}_{\text{var}} &= 0.5
\end{align*}
\]

Using this method, problem instances are generated with only a small number of available units per good, namely 1, ..., 5. To make the sets more comparable, each generated set of instances has the same number of units, see 1.b). The average price per good is between 25 and 75. That is, each unit of that good is worth the respective value. This value later on determines the price paid for a bundle. With step 3, the generation of the bids themselves start. The number of goods in a bid is determined by a decay distribution. At least one good is included in a bid automatically, another one with probability 0.8, etc.. Hence, a bid includes \( i \) goods, \( 1 < i < m \) with probability \( 0.8^{(i-2)} \times 0.2 \), and \( m \) goods with prob. \( 0.8^{(m-1)} \). When a good is included in a bid, the number of units must be determined. This number is, as well, dependent on a decay distribution, this time with probability 0.65. Finally, the price of the bid is the value of the units included in that bid times a uniform random value drawn from \([0.5, 1.5]\).

Leyton-Brown et al. account for their choice as follows. They believe that a small number of goods per bid is a realistic scenario. They point out the independence of this number from the total number of goods. They state that this independence hampers the performance of their branch and bound algorithm, CAMUS. The natural choice of a uniformly drawn number of goods from the available range would, apparently, be much easier for their algorithm. The same argument holds for the small number of units per good. The prices increase linearly in the number of units for a fixed good. This approach, again, seems to guarantee harder instances than other pricing methods (Leyton-Brown et al., 2000). Nevertheless, this linear pricing model conflicts with the notion of synergies. In CAs usually prices are super-linear or sub-linear in the number of units as the bidders
5.1 Test Instances

are expected to value bundles more or less than single units.

The LB instances are now examined with the mentioned indicators for hardness. Following Leyton-Brown et al. we generated 10 test instances with 10 goods and 1500 bids, as well as 10 instances with 14 goods and 2500 bids. To determine the edge density of the BG, the number of edges in the graph must be determined. An edge exists between two nodes, when they are mutually exclusive. In the multi-unit case, they together must desire more units of a good than available. First, let us consider the probability for two bids having at least one good in common. Let X be the amount of goods two bids have in common, that is we search \( P(X \geq 1) \). To determine this probability, we compute

\[
P(X = 0) = \sum_{i=1}^{m-1} \sum_{j=1}^{i} \frac{\prod_{k=1}^{i-1} \left(1 - \prod_{l=1}^{j-1} \right)}{\prod_{k=1}^{i} \left(1 - \prod_{l=1}^{j} \right)} \frac{\prod_{k=1}^{i} \left(1 - \prod_{l=1}^{j} \right)}{\prod_{k=1}^{i} \left(1 - \prod_{l=1}^{j} \right)} \frac{i! \ (m-j)!}{(i-j)! \ m!}.
\]

(5.2)

Both sums list all possibilities for two bids having no good in common. The term \( \prod_{k=1}^{i} \left(1 - \prod_{l=1}^{j} \right) \) denotes the same parameter as in the data generation description. In the settings its value is 0.8. The term \( \prod_{k=1}^{i} \left(1 - \prod_{l=1}^{j} \right) \) stands for the probability that exactly \((m-i)\) goods are included in that bid, \( \prod_{k=1}^{i} \left(1 - \prod_{l=1}^{j} \right) \) denotes the probability that \((j-1)\) goods are chosen in bid2. Remember that one good is included in every bid by default. That is why index \( j \) starts with 1 and index \( i \) is at most \((m-1)\). The index \( i \) reflects the number of goods in bid1 which are not desired. The index \( j \) stands for the number of goods in bid2 which are desired. As example, take a scenario with 10 goods. If \( i = 1 \), then only one single good is not included in bid1. In this case, \( j \) can only take the value 1, since we want both bids to have no bid in common. Bid2 must exactly include the one good, bid1 does not take. Otherwise, say \( i \) takes the maximum value of 9. That is, only one good is included in bid1. Consequently we have to determine the probabilities for all 9 cases, where bid2 includes 1 up to 9 goods. The last term, \( \prod_{k=1}^{i} \left(1 - \prod_{l=1}^{j} \right) \), computes the probability that for the given number of included goods in both bids, they have no good in common. Given that \((m-i)\) goods are included in bid1 and \( j \) goods are included in bid2, the term can be computed by dividing the number of combinations in which both bids have no bid in common by the total number of possible combinations. The number of cases in which both bids have no bid in common is

\[
num_{noCommon} = \frac{m!}{(m-j)! (m-j-(m-i))!} = \frac{m!}{(i-j)!}.
\]

(5.3)

while the total number of combinations is

\[
num_{total} = \frac{m!}{(m-j)! (m-(m-i))!}.
\]

(5.4)

With the help of equation 5.2 we now determine the probability that two bids have a common good. In case of 10 goods, the resulting probability is 74.57%, in case of 14 goods it is 65.69%. Yet, this probability does not describe the event of an edge between two nodes in the bid graph. To that end, the desired number of units must exceed the number of available units for this good. In average the maximal available amount is 3
units per good. In a bid, with \((1 - 0.35) \times 100 = 65\%\) probability, at least two units are desired in a good. Multiplying all these probabilities for the event that two bids are connected with an edge, the resulting probability is roughly spoken between 0.42 and 0.49. Consequently, the BG edge density is high. The clustering coefficient depends only on the edge density. Since goods and units included in bids are determined totally independent, there is no correlation between certain sets of bids. The BGG is sparse, since the number of goods per bid is quite small. The tightness ratio for the instances were determined empirically to be in average 0.23\% for the small instances with 10 goods and 1500 bids and 0.18\% for the larger instances. The edge density and the tightness ratio indicate that the instances are very restricted. Only few bids can be accepted and the inclusion of one bid can already exclude other bids. The low BGG good degree, however, makes the instances less complicated. The constraint matrix is only sparse. It will be interesting to see, which characteristics affect the hardness more.

Before the next type of test data is discussed, we would like to draw attention to the method prices of bids are determined. Unfortunately, the influence of the pricing model is often overseen, although, the similarity between the price determination and the applied bid-sorting heuristic is obvious. Take, for example, the LB method. Here, the prices depend linearly on the value of a good. This factor reminds slightly of the denominator of the SNBP heuristic and strongly of the denominator of the SS heuristic, see equations 4.7 and 4.10. In other words, many algorithms use a heuristic to solve problems whose objective functions were generated with a similar criterion to the one the heuristic uses. This aspect is of great interest and will be discussed in detail in the following analysis. For this purpose, 10 more test instances have been generated to investigate how the pricing method affects the performance of heuristic optimization methods. This time, the prices are assigned randomly between 1 and 1000, independently from any valuations in the bid. The number of goods is again chosen to be 10 and the number of bids to be 1500.

To sum up, with the proposed data generation method by Leyton-Brown et al. (2000b) three different types of test instances were created with 10 instances per set. The first two sets follow strictly the proposed method, including respectively 10 goods and 1500 bids, and 14 goods and 2500 bids. The generation of the third set deviates slightly from the proposed method in assigning prices randomly to the bids. All test instances are highly restricted, but with sparse constraint matrices.

The second type of test instances used in this work is a variance of the PATH distribution from the CATS test suite. To adopt this single-unit generation method, we have programmed a multi-unit version of the PATH distribution. The PATH distribution models a number of partly connected cities. In the generation method a uniform demand for shipping between any pair of cities is assumed. The graph which determines the map is generated with a given number of edges, \(\text{num\_edges}\), being the number of goods, and a given \(\text{edge\_density}\). The number of cities results from the fraction \(\frac{\text{num\_edges}}{\text{edge\_density}}\). Having modelled the map for the scenario, the bids must be generated. It is assumed that each agent is interested in a connection between two cities and demands paths between them. The path can consist of several edges connecting the two cities. All possible paths are separated into different XOR bids, being mutually exclusive. The XOR connectivity is reached by including dummy bids. In this way, it is ensured that an agent only ends up with one possible path between the two cities she wants to connect. The number of edges
5.1 Test Instances

per path is bounded by the parameter $\text{max\_bid\_set\_size}$. On each profit-making path the agent bids a price equal to the utility minus the costs. The costs of shipping from one city to another are equal to the Euclidean distance of the two cities. The utility is random in proportion to this Euclidean distance. The pricing of the bids is similar as in the LB test sets. The prices are linearly dependent on the real value of units in the bid, influenced by a random factor. That is why the same observations concerning prices, made in the section of LB test instances, hold for this kind of instances.

To change the resulting auction to a multi-unit CA, we assigned a uniformly distributed number of $\text{max\_units}$ to each edge in the range $[1,10]$. In the bid generation method a number of units is determined uniformly in the range of $[1,\text{max\_units}]$ for each edge which is desired. As input parameters the default values from the CATS distribution are taken with an $\text{edge\_density}$ of 3 and a $\text{max\_bid\_set\_size}$ of 5. We have chosen 50 goods and 1500 bids. Yet, since the algorithm includes dummy goods automatically, additionally to the 50 bids, between 443 to 664 dummy goods are created. It makes the problem more restricted, since there is only one single unit available per dummy good. The reason, why we do not deviate from the default parameters is to first concentrate on standard test instances from the two problem domains. It would go beyond the scope of this work to test the algorithms on all possible parameter settings. Yet, after the analysis we are able to make judgements about these standard test instances and their influence on the performance. With the so gained knowledge, a selective, systematic choice of parameters should be possible in future.

Since the number of goods in the MULTIPATH instances is much larger than the number in the LB instances, the hardness of the MULTIPATH test instances should differ from the LB instances. On the one hand, it will be even more unlikely that two bids are mutually exclusive. On the other hand, the dummy goods artificially make all bids of the same agent mutual exclusive and add a lot of edges to the BG graph. In contrast to the LB instances, the clustering coefficient, reflecting the number of edges among neighbors, is supposed to be quite high. That is because in a bid one path leading from one city to another is included. Therefore neighborly paths have a higher probability to appear in neighbored bids since they bid on the same or, at least, similar edges. The tightness ratio is with an average of 5.1% larger than in the LB instances, but still quite low. The low ratio is caused by the small number of goods per bid and the included dummy goods. The BGG good degree is even lower than in the LB instances because the number of goods per bid is very low, and in total more goods are available. To sum up, in the MULTIPATH test set a lot of goods are available, the structural properties indicate a rather highly restricted problem, and the matrix is very sparse.

Obviously, instances in the MUCA literature all represent auction scenarios where only a small ratio of the available goods are included in a bid. We doubt that the focus on such scenarios is the appropriate approach. Since data from real-world applications is rare, experience how possible scenarios might look like is almost nonexistent. Hence, in our view, it is the task of research to deal with a diverse amount of test instances. In this way, it can be assured that the investigated algorithms can be applied in various upcoming real-world auction scenarios. Researchers might argue that although real data is almost nonexistent, one can still imagine scenarios which might be relevant in future. If so, we don’t agree that exclusively scenarios should become relevant, in which only a small
number of goods is included per bid. First of all, a CA is only appropriate for scenarios where there are synergies among goods. Consequently it is much more realistic to assume the bidders to bid on several goods. The synergies are stronger the more different goods are included in the bundle. Second, with further research on other complexity problems of CAs, like communication complexity, it will become easier for agents to submit bids with a larger amount of goods included. It can be assumed that algorithms will help the bidders to formulate complicated bids which fully express their utilities, including sophisticated synergies. Therefore, a trend to more complex bidding with more goods and units included in bids is most probable. Last but not least, there are many concrete scenarios, where a dense constraint matrix is very probable. For example, in bandwidth allocation problems when providers want to offer service for all connections in the network. In general, in all logistic problems service providers might not concentrate on separate regions but aspire a full cover of the network. Another example are procurement auctions in industries where a large number of raw material or preliminary products is needed by every company producing a common product type. The listed arguments motivate to search either for existing data with denser matrices, or to develop new data generation methods with this property. We decided for the first approach as the test instances from the MDKP literature fulfill exactly this requirement.

The MKNAPCB test instances are taken from the OR-library. They are the standard instances the most relevant algorithms used for comparison (Gottlieb, 1999a, Raidl, 1999, Chu and Beasley, 1998). Overall, Chu and Beasley (1998) generated 30 instances for each of the 9 combinations \( m = \{5, 10, 30\}, \ n = \{100, 250, 500\}^3 \), resulting in a total of 270 instances. Since the tightness ratio is a measurement of restriction in the MDKP literature, instances were created with a specified tightness ratio. For this purpose, first, for all \( i = 1, ..., m \) and \( j = 1, ..., n \) the resource consumptions \( r_{ij} \) are drawn uniformly from the range \([0, 1000] \). From this it follows that all items consume capacity of almost all available resources, causing the desired dense constraint matrix with very few \( r_{ij} = 0 \).

For each of the 9 combinations 10 instances were created with \( \alpha = 0.25, \ 0.5, \ 0.75, \) respectively. That means that \( \alpha \) of the desired consumption is available. The price of an item is determined as follows:

\[
p_j = \sum_{i=1}^{m} \frac{r_{ij}}{m} + 500 \ R[0, 1],
\]

where \( R \) signifies a real number drawn from a uniform distribution. This pricing method differs from the already seen pricing methods. The value of all goods is considered to be equal. The fraction just computes the average amount of units per good desired in bid \( j \). In expectation, it is supposed to be about 500. This fraction is equal to the one used by the NBP heuristic. Consequently, it will be interesting to observe the performance of the NBP heuristic compared with others. The number of dimensions (goods) is comparable with the number in the test instances generated with the MUCA methods, but the number of bids (items) is much smaller. Furthermore, the constraint matrix is much denser than the ones in the MUCA problems because all bids include all available goods. Concerning the BG graph, at first glance one might expect a high edge density, because all bids include

\(^{3}\)As usual \( m \) denotes the number of dimensions (goods) and \( n \) the number of items (bids).
almost all goods. Then, however, due to the high tightness ratio a lot more resources than in the LB and in the MULTIPATH case are available. Thus, it is very improbable that two bids request all available units of a good. In most cases, the BG graph will be fully unconnected and the edge density will be 0. To sum up, MKNAPCB instances have very dense constraint matrices. The high tightness ratio, the very low edge density, and the resulting low clustering coefficient indicate a weakly restricted problem.

5.2 Performance

In the following section, different algorithms for solving the MDKP and the multi-unit WDP will be presented. Their performance on the just discussed test instances will be described and analyzed. The first two approaches are optimal algorithms. With the given computational capacities in the experiments, they run out of memory and time when test instances have certain properties. That is why, afterwards, we will consider heuristic optimization methods. First, the performance of primal greedy heuristics will be analyzed. They are important, since they bias the search in most meta-heuristics; in particular, in the finally presented weight-coded approach by Raidl (1999).

5.2.1 CAMUS

The branch and bound algorithm, CAMUS, is presented in section 3.1.1. Since it is an optimal algorithm, its running time is taken as performance criteria. The experiments were made on an Intel Xeon with 3 GHz and 2 GB RAM. The source code was written in C++ and is available in the internet by courtesy of Kevin Leyton-Brown\(^4\). The experimental results are listed in table 5.2.

<table>
<thead>
<tr>
<th>test data</th>
<th>m</th>
<th>n</th>
<th>time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MKNAPCB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>all 270 instances</td>
<td>10</td>
<td>1500</td>
<td>0.062</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1500</td>
<td>3.953</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1500</td>
<td>0.047</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1500</td>
<td>0.219</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1500</td>
<td>0.500</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1500</td>
<td>0.047</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>1500</td>
<td>0.063</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1500</td>
<td>0.047</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>1500</td>
<td>0.047</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1500</td>
<td>0.235</td>
</tr>
<tr>
<td>average</td>
<td>10</td>
<td>1500</td>
<td>0.522 ((\sigma = 1.21))</td>
</tr>
</tbody>
</table>

Table 5.2: Performance of CAMUS. (continued)

<table>
<thead>
<tr>
<th>test data</th>
<th>m</th>
<th>n</th>
<th>time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>random_1</td>
<td>10</td>
<td>1500</td>
<td>0.063</td>
</tr>
<tr>
<td>random_2</td>
<td>10</td>
<td>1500</td>
<td>0.172</td>
</tr>
<tr>
<td>random_3</td>
<td>10</td>
<td>1500</td>
<td>0.078</td>
</tr>
<tr>
<td>random_4</td>
<td>10</td>
<td>1500</td>
<td>0.062</td>
</tr>
<tr>
<td>random_5</td>
<td>10</td>
<td>1500</td>
<td>0.093</td>
</tr>
<tr>
<td>random_6</td>
<td>10</td>
<td>1500</td>
<td>0.187</td>
</tr>
<tr>
<td>random_7</td>
<td>10</td>
<td>1500</td>
<td>0.156</td>
</tr>
<tr>
<td>random_8</td>
<td>10</td>
<td>1500</td>
<td>0.078</td>
</tr>
<tr>
<td>random_9</td>
<td>10</td>
<td>1500</td>
<td>0.063</td>
</tr>
<tr>
<td>random_10</td>
<td>10</td>
<td>1500</td>
<td>0.047</td>
</tr>
<tr>
<td>average</td>
<td>10</td>
<td>1500</td>
<td>0.099 (σ = 0.05)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>14</th>
<th>2500</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>2500</td>
<td>6.265</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>2500</td>
<td>2.203</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>2500</td>
<td>24.187</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>2500</td>
<td>21.438</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>2500</td>
<td>13.140</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>2500</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>2500</td>
<td>327.779</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td>2500</td>
<td>12.860</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>2500</td>
<td>47.014</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>2500</td>
<td>1.141</td>
</tr>
<tr>
<td>average</td>
<td>14</td>
<td>2500</td>
<td>50.670 (σ = 104.86)</td>
</tr>
</tbody>
</table>

**MULTIPATH**

<table>
<thead>
<tr>
<th></th>
<th>595</th>
<th>1500</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>average</td>
<td>595</td>
<td>1500</td>
<td>-</td>
</tr>
</tbody>
</table>

According to the IP formulation of the problem, \( m \) denotes the number of goods or the number of dimensions, while \( n \) denotes the number of bids or items. The MULTIPATH instances were generated with 50 as input for the number of goods, but the algorithm automatically adds a lot of dummy goods to create XOR bids. That is why the number of goods is much larger, in average 595. The three types of test data MKNAPCB, LB, and MULTIPATH are printed in bold as headings of the respective rows. Each row in the LB section represents one instance. The average of each parameter setting is printed together with the standard deviation for the measured running times at the end of each block.

CAMUS aborts for all 270 MKNAPCB instances. Probably the internal data structure the implementation uses is inappropriate to solve problems with such a large number of units; already in the smallest instance there are around 12500 units per good available. The instances of the LB test data are the ones CAMUS was also originally tested with. They are solved quite reliably. Particularly, the random instances are solved very fast. This excellent performance is emphasized by the statement in Leyton Brown et al. (2000b) that CAMUS has much more problems with linearly dependent prices than with random ones. To compute the larger instances with 14 goods, the algorithm requires 10 times
longer than to solve the smaller ones. Also the standard deviation grows with the problem difficulty. Especially in the case with 14 goods, the deviation is very high, the times range from about one second to more than five minutes. The sixth instance from the set with 14 goods causes problems. Although the optimal solution is found after only 1.39 seconds, the algorithm has problems to prove its optimality. Thus, it was stopped after 10 hours. The 10 MULTIPATH instances were all stopped after 10 hours running time with a gap between 10% and 20% to the optimum.

To sum up, CAMUS works well for the test instances which were originally created for its testing. Apparently, it has huge problems once the number of goods and units increases. Yet, without results for the MKNAPCB and MULTIPATH problems, it is unclear if maybe other structural properties cause the bad performance. It might be the case that CAMUS has problems with the high tightness ratio of the MKNAPCB instances. The very low BG edge density might make the pruning hard. In the next section we point out that CPLEX has improved drastically since the development of CAMUS. It outperforms CAMUS by several orders of magnitude. That is why a further observation of CAMUS is not necessary.

5.2.2 CPLEX

Experiments with ILOG CPLEX 9.0 were conducted on a Linux machine, AMD Opteron, 64-Bit, 2.2 GHz, and 2 GB RAM. The results are shown in table 5.3. The table differs slightly from the one for CAMUS. All problem instances are consolidated according to the parameter settings, while for the CAMUS table we listed all instances from the LB data set separately. Each line in the table for the MKNAPCB instances refers to 30 test instances, the rest refers to 10 instances. The denoted times reflect the average time for one instance from the respective parameter group. In case CPLEX runs out of memory (out of mem.), we have listed the times when CPLEX aborted. The gap in the last column refers to the best solutions reported in the OR-library. For all test instances which were not taken from this library this gap cannot be computed. All optima found by CPLEX which are not specified in the data of the OR-library can be found in the appendix in table A.1.

Table 5.3: Performance of CPLEX. The 1-10* instances have randomized prices.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>optimum found</th>
<th>time (sec.)</th>
<th>gap[%]</th>
<th>1-(OR-lib./CPLEX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>yes</td>
<td>224</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>yes</td>
<td>3084</td>
<td>0.003</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>yes</td>
<td>3840</td>
<td>0.040</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>yes</td>
<td>378</td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>out of mem.</td>
<td>322</td>
<td>failed</td>
<td></td>
</tr>
</tbody>
</table>

Unfortunately the runs for CAMUS and CPLEX could not be made on the same machine, as for CAMUS we have a Windows executable, and the CPLEX licence we have access to is installed on a Linux machine.
Table 5.3: Performance of CPLEX. (continued)

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>optimum found</th>
<th>time (sec.)</th>
<th>gap [%]</th>
<th>1-(OR-lib./CPLEX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>10</td>
<td>250 out of mem.</td>
<td>820</td>
<td>failed</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>100 yes</td>
<td>1152</td>
<td>0.018</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td>250 out of mem.</td>
<td>608</td>
<td>failed</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td>500 out of mem.</td>
<td>266</td>
<td>failed</td>
<td></td>
</tr>
</tbody>
</table>

**LB**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
<td>10</td>
<td>1500 yes</td>
<td>1.2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1-10*</td>
<td>10</td>
<td>1500 yes</td>
<td>1.4</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1-10</td>
<td>14</td>
<td>2500 yes</td>
<td>1.6</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

**MULTIPATH**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
<td>595</td>
<td>1500 yes</td>
<td>9.1</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

The table shows that only MKNAPCB instances with 100 items or 5 dimensions can be solved optimally. For the cases CPLEX did not abort, the running times increase drastically with the number of items and the number of dimensions. For test instances 3 with 5 dimensions and 500 items, CPLEX takes, in average, already 64 minutes to solve one instance. Obviously, CPLEX solves all LB and MULTIPATH instances very fast to the optimum. However, the increased number of 14 goods only causes an increase of 133% in running time, which is much less than the 1000% increase by CAMUS.

Chu and Beasley (1998) also tried to solve the MKNAPCB instances with the CPLEX mixed-integer programming solver, version 4.0. At that time, CPLEX succeeded in solving only test data sets 1, 2, and 4. Apparently, the optima of these CPLEX runs are not included in the information about the best known solutions reported in the OR-library. Otherwise, the gap of MKNAPCB 1, 2, and 4 would be 0.

The experiments show that CPLEX is a very good choice for solving test instances, as found in the CA literature. However, it runs into memory and time problems with larger instances having a dense constraint matrix and little restrictions. If only the former instances were relevant for WDPs, there would be no need to develop and analyze heuristic optimization methods. CPLEX seems to solve them to perfect satisfaction. Nevertheless, in this work, the question was raised whether these instances are appropriate for future real-world applications. We argue that the instances taken from the MDKP literature might be relevant, as well, for MUCA problems. The dense constraint matrix reflects many kinds of problems occurring in real-world auction scenarios. Moreover, the most difficult test instances with 30 goods and 500 bids seem not to be oversized for e-auctions. As CPLEX is not able to solve the larger MKNAPCB instances optimally, heuristic optimization methods are appropriate to deal with them. Therefore, when analyzing the weight-coded approach, the focus is on the MKNAPCB instances.

The excellent performance of CPLEX for the MUCA instances might be due to their strict constraints. CPLEX can exclude a lot of combinations, when a decision variable is assigned to one, leading to a search in a small region. This makes it probably very fast and easy to prove the optimal solution. Yet, in the work by Leyton-Brown et al. (2006),
experiments with \{1000, 2000\} bids and \{64, 144, 256\} goods show that more restrictions make test instances harder for CPLEX. They argue that higher constraints might cause more costs than savings. The savings come from searching in a smaller search space. The increased costs for highly restricted problems result either from an increased complexity for each calculation on a node, or from a decreased accuracy of the LP relaxation. We think that in case of the LB and MULTIPATH instances, the advantage of the small search space still exceeds the costs. This is because, compared to Leyton-Brown et al., we have chosen a much smaller number of goods and bids.

The MKNAPCB instances are much less restricted than the MUCA instances, but the constraint matrix has almost no zero entries. Probably the number of non-zero entries is responsible for the bad performance of CPLEX. The denser the matrix, the more complex is the problem. Therefore, with roughly more than 2500 non-zero entries and more than 250 bids, CPLEX is no longer able to compute the optimum. The latter probably arises from the increased computational complexity at each node, already alluded to.

Finally, a dependence on the tightness ratio is observed. Among the problems CPLEX solves, the ones with a lower tightness ratio are more time-consuming. For set 4, for example, CPLEX took 1.48 times as long for the instances with \(\alpha = 0.25\) compared to the ones with \(\alpha = 0.50\), the ratio \(\alpha = 0.50\) to \(\alpha = 0.75\) was 3.45. The same trend has been observed with other instances. It supports the already mentioned statements by Leyton-Brown et al. concerning cost and savings of highly constricted problems. To sum up, if the instances have low complexity, measured with the BGG edge degree, and accordingly with the constraint matrix density, highly restricted problems can still be solved very fast. If, however, there is a high complexity, highly restricted problems are hard for CPLEX. Furthermore, larger problem instances with a dense matrix cannot be solved by CPLEX at all with the given computational capacities.

### 5.2.3 Primal Greedy Heuristics

In the last sections, the necessity to apply non-optimal algorithms to more complex test instances was pointed out. As shown in chapter 4 most non-optimal algorithms use some kind of problem-specific bias. Meta-heuristics, the focus of this work, include different bid-sorting heuristics for biasing the search, presented in section 4.3. In our view, it is crucial to first examine the performance of these primal greedy heuristics before incorporating them in more complex meta-heuristics. Since the effect of structural properties of test instances on the performance is complex, a complete analysis is beyond the scope of this work. Therefore, we concentrate on the influence caused by the methods for generating prices. This is a crucial aspect which should be considered in future artificial data generation methods.

In table 5.4 the performance of the four primal greedy heuristics NBP, SNBP, RLPS, and SS are illustrated. As the heuristics are deterministic, one run per instance is sufficient. The performance is measured with two gaps in percent. \(\text{GapR}\) refers to the objective value of the relaxed LP, as defined in section 3.1, while \(\text{gapB}\) refers to the optimal solution to the original LP with binary variables. Since the optimal values of some of the MKNAPCB instances are unknown, some entries in the \(\text{gapB}\) column are empty. As the relaxed LP solution is an upper bound for the binary LP, the \(\text{gapR}\) is always larger than the \(\text{gapB}\). In contrast to the previous tables, the MKNAPCB instances are listed in more detail. Each
set is divided in subsets of instances with the same tightness ratio $\alpha$. Thus, each row in the table represents 10 instances.

Table 5.4: Performance of primal greedy heuristics in %. The * instances have randomized prices.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>$\alpha$</th>
<th>NBP[%]</th>
<th>SNBP[%]</th>
<th>RLPS[%]</th>
<th>SS[%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>gapR</td>
<td>gapB</td>
<td>gapR</td>
<td>gapB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| MKNAPCB
| 5  |  100 |  .25    | 9.611  | 8.706  | 7.453  | 6.525 |
|    |      |  .50    | 4.595  | 4.163  | 3.203  | 2.764 |
|    |      |  .75    | 2.696  | 2.386  | 1.695  | 1.381 |
|    |  250 |  .25    | 7.515  | 7.310  | 4.657  | 4.446 |
|    |      |  .50    | 4.126  | 4.022  | 2.148  | 2.108 |
|    |      |  .75    | 2.192  | 2.117  | 1.009  | 0.933 |
|    |  500 |  .25    | 7.012  | 6.945  | 3.170  | 3.011 |
|    |      |  .50    | 3.626  | 3.587  | 2.148  | 2.108 |
|    |      |  .75    | 2.247  | 2.221  | 1.115  | 1.089 |
| 10 |  100 |  .25    | 11.099 | 9.687  | 8.959  | 7.513 |
|    |      |  .50    | 5.255  | 4.500  | 4.029  | 3.263 |
|    |      |  .75    | 2.821  | 2.350  | 3.086  | 2.616 |
|    |  250 |  .25    | 7.912  | 6.005  | -      | -     |
|    |      |  .50    | 3.952  | 3.435  | -      | -     |
|    |      |  .75    | 1.852  | 1.575  | -      | -     |
|    |  500 |  .25    | 6.367  | 3.524  | -      | -     |
|    |      |  .50    | 3.27   | 2.347  | -      | -     |
|    |      |  .75    | 1.636  | 1.248  | -      | -     |
| 30 |  100 |  .25    | 11.394 | 8.758  | 11.526 | 8.896 |
|    |      |  .50    | 6.431  | 5.179  | 5.975  | 4.717 |
|    |      |  .75    | 3.480  | 2.678  | 3.318  | 2.514 |
|    |  250 |  .25    | 7.899  | 7.186  | -      | -     |
|    |      |  .50    | 5.001  | 4.206  | -      | -     |
|    |      |  .75    | 2.267  | 2.214  | -      | -     |
|    |  500 |  .25    | 6.972  | 5.515  | -      | -     |
|    |      |  .50    | 3.199  | 2.829  | -      | -     |
|    |      |  .75    | 1.624  | 1.542  | -      | -     |
|    |      | average | 5.039  | 4.974  | 3.897  | 3.592 |
| LB
| 10 | 1500 |  .0023  | 5.170  | 2.844  | 20.221 | 18.287 |
|    |      | *10     | 1.103  | 0.945  | 0.357  | 0.199 |
| MULTIPATH
| 595| 1500 |  .05   | 9.773  | 6.328  | 19.222 | 16.131 |

average | 5.039 | 4.974 | 3.897 | 3.592 |

LB
| 10   | 1500 |  .0023  | 5.170  | 2.844  | 20.221 | 18.287 |
|      |      | *10     | 1.103  | 0.945  | 0.357  | 0.199 |

MULTIPATH
| 595 | 1500 |  .05  | 9.773  | 6.328  | 19.222 | 16.131 |

average | 5.039 | 4.974 | 3.897 | 3.592 |
The most obvious information the table provides is that all presented primal greedy heuristics provide very good results. For the MKNAPCB the gaps are in the range from 0.116% to 11.524%, the \( \text{gap}_B \) even stays constantly below 10%. For the MUCA instances the gaps are higher, but the worst gap is still only 18.287%. For the three types the heuristics perform differently. The RLPS greedy heuristic is the best one for the MKNAPCB instances with 0.931% \( \text{gap}_B \), followed by the SS heuristic with 1.560%. For the LB instances there is no common winner, and for the MULTIPATH instances it is RLPS again. Regarding the LB test data, some heuristics worsen for the instances with more goods and bids, some improve. However, they are all very good for the data with randomized prices. The results for the MULTIPATH instances are among the worst ones.

Concerning the MKNAPCB instances, mainly three trends are observable. First, when the number of goods augment, the gaps increase. Second, when the number of bids augment, the gaps decrease. Third, with increasing tightness ratio all heuristics improve, see figure 5.1. An observation very similar to the third one was made by Raidl and Gottlieb (2005) in their recent work. In contrast to our work, they only consider the gap to the relaxed LP. They state that, generally, one cannot conclude that the heuristics are better with larger tightness ratios. They reason that this trend is caused by the tighter bound of the relaxed LP for higher tightness ratios. However, we can clearly infer from our experiments that also the \( \text{gap}_B \) diminishes. Hence, there must be a dependency between the performance of the primal greedy heuristics and the tightness ratio.

<table>
<thead>
<tr>
<th>input</th>
<th>gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>goods</td>
<td>↑</td>
</tr>
<tr>
<td>bids</td>
<td>↓</td>
</tr>
<tr>
<td>α</td>
<td>↓</td>
</tr>
</tbody>
</table>

Figure 5.1: Influence of input sizes on performance of primal greedy heuristics.

In the following paragraphs we first discuss the three trends found for the MKNAPCB instances. After this, the influence of structural properties of the three types of instances is examined with focus on the pricing model. As just mentioned, the gap of the relaxed LP also varies with different structural properties, like the tightness ratio. This variation can lead to wrong conclusions as two effects influencing the gap might interfere. That is why we focus the analysis of this section only on the \( \text{gap}_B \).

**Influence of goods on performance** Only the cases with 100 bids can be compared being the only ones where the \( \text{gap}_B \) is known for different number of goods. Consider, for example, the SNBP heuristic. For \( \alpha = 0.25 \) and \( \alpha = 0.50 \) the gap increases constantly from 6.525% to 8.896% and from 2.764% to 4.717%. For the highest tightness ratio, the behavior is not as constant. Yet, for the RLPS and the SS heuristic, the dependency is even stronger and can be observed for all tightness ratios. We think that two effects cause the increase. One holds for all four heuristics and the other one is only true for the SS
and the RLPS heuristics. That is why the reaction of the SS and the RLPS heuristic is amplified. We imagine the reason which holds for all bid-sorting heuristics to be the following. With an increased number of goods, the number of inequalities describing the constraints raises. It complicates the process of finding the optimum. Through acceptance of only one bid, many other bids can be excluded. If only one bid has a position too early in the front of the priority queue, the solutions from the heuristic and the optimal solution can differ a lot. This is because the misleading acceptance of a bid, which is not included in the optimal solution, has more effect on the other, unaccepted bid, if the problem is highly restricted. Therefore, the gap increases with an augmenting number of goods. The second effect, which only concerns the RLPS and the SS heuristic, could be explained as follows. Considering the gaps to the optimum solutions for a fixed number of 100 bids and $\alpha = 0.25$, the RLPS develops from 1.478% to 2.082% to 3.306%. The deterioration of performance is even harder in case of the SS heuristic with 1.557% to 3.182% to 7.215%. The cause for the deterioration is that the solutions to the relaxed LP dissociate from the solutions to the binary LP with increasing number of goods. Since RLPS and SS are dependent on the performance of the relaxed LP, they follow this trend.

**Influence of bids on performance** Unfortunately, only the gapB in case of five goods can be compared. Here again, the effect is stronger for the SS and RLPS heuristic. Once more, we see the performance of the relaxed LP as reason for the overlaying effect. Note that in the relaxed LP only up to $m$ variables are fractional. All other ones are either 0 or 1. When the number $n$ of decision variables increases (bids), and the number of goods $m$ stays fixed, the proportion of fraction variables, $\frac{m}{n}$, decreases. Therefore, the ratio of variables set to 0 and 1 gets larger. Thus, the solution to the relaxed LP only differs in a small amount of variables from the binary solution which might lead to a similar objective value. The cause for the dependency holding for all four heuristics is not very obvious. Thus, we only have an idea which might be responsible for the given phenomena. Imagine a bid-sorting heuristic traversing the prioritized list of bids, checking each bid for inclusion. In the case where only few bids are available, the heuristic has a narrowed selection of remaining bids when already most capacities are consumed. Therefore, it will be improbable that one of the few bids remaining still fits in the allocation. In contrast, in the case with more bids the probability increases leading to a higher expectation of the objective value.

**Influence of tightness ratio on performance** Once the problem is less constraint, the primal greedy heuristics are more successful. Take for example the first set of MKNAPCB instances and the NBP heuristic. The gapB almost halves with increasing $\alpha$, from 8.706% to 4.163% to 2.386%. When discussing the influence of goods, an idea why with less constraints the heuristics are better, was already given. Nevertheless, we would like to observe the influence of the tightness ratio in more detail. It is equiprobable to guess the exact solution with an $\alpha = 0.25$ and an $\alpha$ of 0.75, since $\binom{100}{25} = \binom{100}{75}$. But, in this case, not the exact guessing is relevant, but how well the optimum is approximated. We suppose that the higher number of accepted bids for higher tightness ratios is responsible for the effect. Imagine a string with four bids. With an $\alpha = 0.25$ about one bid can be accepted, with an $\alpha = 0.75$ already about three bids can be accepted. In the first case,
there are three other possibilities to place the 1 so that the strings have no 1 in common. As an example, say the optimum is [1000], we can create the 3 strings [0100],[0010], and [0001]. Since, the prices of bids usually differ, the three alternative strings will lead to very different objective values. Contrariwise, with a tightness ratio of 0.75, there is no possibility to place the ones so that totally different bids are accepted. As an example, take [1110] as optimum. The objective value of every alternative string with three ones and the same length will only differ slightly from the optimum, since two ones are always in common. This small example can be generalized to a larger number of bids and shows that the probability to have a close to the optimum objective value is much higher in scenarios where a lot of bids are accepted. The reasons for the conclusion can be summed up as follows. The same Hamming distance between pairs of strings with a different number of ones accepted, leads to more changes in the value of the objective function in the case with less accepted bids. We see this as partly explanation for the primal greedy heuristics performing better for higher α.

Influence of pricing model on performance  Last objective of the analysis is to examine why the performance differs in dependence on the type of test instances. The basic difference between NBP and the two heuristics SNBP and SS is that in NBP all goods are treated equally. In SNBP and SS there is a relevance value multiplied with the resource consumptions to express the scarcity of a capacity. If the bid consumes a lot of scarce resources the heuristic assigns a lower value to the bid. Therefore, the goods have different valuations. The interpretability of the RLPS heuristic is not easy, as it is hard to comprehend what happens internally in the relaxed LP. That is why we mostly focus on the other three heuristics.

The cause for the dependency is probably complex. Yet, the very good performance for the LB test instances with randomized prices indicates that the pricing model plays a major role since this set was, aside of the different pricing method, created with exactly the same parameters as the set with 10 goods, one row above in the table. All heuristics compute these instances with a very small gap, even the worst one by NBP is only 0.945%. Additionally, it is surprising that NBP is the best heuristics for the standard instance with 10 goods and SS is the best one in the case with randomized prices. Thus, the performance of the heuristic has to be strongly dependent on the pricing method chosen. This means that certain bids have a higher price than they are worth.

The main idea of the following analysis is that the pricing model of the generation methods favor certain bids while penalizing others. We suggest that when certain structures of bids are favored by the pricing model, the heuristics which favor exactly the same structures will perform better. In other words, heuristics need to have a similar bias to certain structural properties of bids than the pricing methods. Therefore, we analyze what bias the pricing method makes, and compare it with the bias made by the heuristic. Furthermore, it is interesting to examine the values according to which the heuristic sort the bids, given the specialized pricing method. This shows what information about the bid is finally used by the heuristic for the sorting process.

With random prices, it is obvious that bids with a high price, but little units or goods included, are the best ones. They consume little resources while contributing a high value to the objective function. This makes it easy for all heuristics which set the price in
5.2 Performance

relation to the size of the content. That is probably why NBP, SNBP, and SS provide very
good results. We think that with randomized prices RLPS might assign a larger number
of ones and only a small number of fractional variables, since for many decision variables it
will be obvious that they should be included in the solution. This would result in a higher
objective value. The more sophisticated the value of the included units is estimated, the
better the heuristic works. In the table, the degree of sophistication for NBP, SNBP, and
SS increases from left to right. The NBP is the most simple heuristic and SS the most
complicated one, leading to the best results for randomized pricing.

In the following paragraphs, we concentrate on the pricing method of MKNAPCB and
LB to demonstrate how they can be analyzed. To begin with, we think about what kind
of bids are favored by the MKNAPCB and the LB pricing methods and which ones are
disfavored. The prices for MKNAPCB instances are determined as follows:

\[ p^M_j = \frac{m}{m} \sum_{i=1}^{n} r_{ij} + 500 \cdot R[0, 1]. \]  

The ones for the LB instances are

\[ p^{LB}_j = R[0.5, 1.5 \cdot \sum_{i=1}^{m} (\text{avgprice}_i \cdot r_{ij}). \]  

The prices for MKNAPCB problems assume equal values for all units. The price is deter-
mined by the number of average units per good and a random number with expectation
250. The random number is added to the first term and is in expectation half as high,
since \( r_{ij} \) is drawn uniformly from \([0,1000]\). Important is the concatenation by addition
which makes the increase by the random number totally independent from the content
of the bid. Therefore, bids with a small amount of units per good, but a high random
number are favored. Although the bid consumes only few units, the price is very high.
In contrast stands \( p^{LB}_j \). Here, the method assumes different values for units, however,
units of the same good have the same price. The influence of the random number is in
expectation higher than in the MKNAPCB case. First, it is multiplicative and second,
the range is larger. It can change the price of the bid from half its value to 1.5 times its
value. Therefore, it favors bids with a large \( \sum_{i=1}^{m} (\text{avgprice}_i \cdot r_{ij}) \).

To sum up, the two methods create the following biases. The MKNAPCB method
assigns relatively high prices for bids which include only a small number of units per good,
but are updated with a high random value. The LB method favors bids with a high
amount of valuable units which are updated with a high random number. In general, the
influence of the random number is higher in the LB case. Now, let us examine whether
the heuristics favor the same bids as the pricing methods. If so, the objective function of
the primal greedy heuristic is supposed to be high. It accepts, mainly, the bids for which
a good price for little consumption is achieved.

The NBP heuristic assigns a value equal to the \( p_j \) divided by \( \sqrt{\sum_{i=1}^{m} r_{ij}} \). The sum over
all \( r_{ij} \) denotes all units included in a bid. Therefore, NBP describes some kind of average
value a unit in the bid has. Yet, the root of the sum biases this average value, since \( \sqrt{x} \) is
a declining function. The slope of the sum diminishes with an increasing number of units.
Because the root is in the denominator, NBP prefers bids with a large number of units,
similar to the LB instances. It is supposed to perform better on those instances than on the MKNAPCB ones.

The SNBP heuristic favors bids paying a high price for units of goods with high capacity. Instead of taking the root of the sum, it computes \( \sum_{i=1}^{m} \frac{r_{ij}}{c_i} \). The less the desired units demand of a good’s capacity, the lower the denominator, and the higher the SNBP. The relevance value \( \frac{1}{c_i} \) is totally uncorrelated to the average price the LB pricing methods assumes. Therefore, we conclude that SNBP favors different bids than the LB instances. This explains its bad performance on those instances. Contrariwise, it favors similar bids than MKNAPCB, since a small number of units per good lead to a high SNBP, which might explain its good performance on the MKNAPCB instances.

Concerning the SS heuristic, general statements are hard as it strongly depends on the value of the dual prices. We investigated that the dual prices of the relaxed LP are much more constant in the MKNAPCB case than in the LB case. This seems reasonable as only in the LB instances different values per good are assumed. We tested this assumption with 50 instances of MKNAPCB and LB, with each 5 goods and 100 bids in the MKNAPCB case and 1500 bids in the LB case. The values seemed normally distributed. All dual prices were normalized to the range \([0,1]\). For the MKNAPCB instances the average was 0.7 with a standard deviation of 0.18, while for the LB instances the average was 0.44 with a much higher standard deviation of 0.28. Yet, these observations are not sufficient for judging the influence of pricing on the performance of the SS heuristic. Further experiments have to be carried out, before a general conclusion can be drawn.

To sum up, we have presented a method to analyze the dependency of performance on the method of generating prices. This approach compared the bias made by the data generation method in favor to certain structural properties of bids and the bias the heuristics make. The more similar the bias is, the more successful is the primal greedy heuristic. To finish, we suggest one other approach to test the dependencies. It was just pointed out according to which criteria the heuristics aim to sort the bids. Knowing the methods how prices were originally determined, the nominators \( p_j \) from the heuristics can be replaced by equations 5.6 and 5.7. The resulting fractions show the real information according to which the heuristics sort the bids. In other words, there is a difference between the intended sorting criteria and the final information which is taken to sort the bids. Consider, for example, the fraction with which NBP orders the bids. Replacing \( p_j \) in the nominator with the real equation for prices in LB instances, results in the following equation:

\[
NBP = \frac{p_j}{\sqrt{\sum_{i=1}^{m} r_{ij}}} = \frac{R[0.5, 1.5] \times \sum_{i=1}^{m} (\text{avgprice}_i \times r_{ij})}{\sqrt{\sum_{i=1}^{m} r_{ij}}}.
\]

The information content of the fraction on the right hand side seems still to reflect the original ideas according to which NBP sorts the bids. It computes the value of one unit in the bid distorted by some random variable and biased by the root function. In contrast consider the MKNAPCB case:

\[
NBP = \frac{1}{m} \sqrt{\sum_{i=1}^{m} r_{ij} + \frac{500 \times R[0, 1]}{\sum_{i=1}^{m} r_{ij}}}.
\]

This fraction is harder to interpret and seems to diverge more from the original sorting criteria. It is unclear if the NBP still uses appropriate information to sort the bids.
5.2 Performance

information content is distorted. Similar analyses could be made with other heuristics. However, it is hard to always judge the degree of distorted informational content. In future work some kind of common concept should be developed to judge the divergence between originally intended sorting criteria and informational content which is finally taken to sort the bids, assumed that the generation method for the prices is known.

In this analysis we have learned that in average the primal greedy heuristics perform well, particularly when taking into account the fast running times of only a few milliseconds. The gap for the MULTIPATH instances is, for instance, better than the one found by CAMUS after 10 hours running time. Consequently, the idea to bias the search process with these heuristics is a very good approach. Furthermore, a dependency between test instances and performance was identified. Concerning the MKNAPCB instances, we examined the influence of the number of goods, the number of bids, and the tightness ratio. In order to select the best heuristic for certain structural properties of test instances, we have concentrated only on price issues. To this end, two methods for analyzing the influence of pricing methods were discussed. Other structural properties, like the density of the constraint matrix and the BG edge density seem to have a more complex influence. A conclusion might be drawn from the good results of heuristics with relevance value (SNBP and SS) in case of the MKNAPCB instances. Since those instances have dense matrices, a different valuation of goods might be very important.

To conclude, the dependency between structural properties of data and heuristics indicates that meta-heuristics should try to apply different bid-sorting heuristics for solving instances. Nevertheless, as long as the dependencies are not totally clear, it is hard to predict which heuristic will work best in advance. Yet, we have provided first steps into this direction. In the following sections, we will investigate whether better improvements of existing meta-heuristics can be made by restarts with different bid-sorting heuristics or by tuning the parameters for a fixed heuristic.

5.2.4 Raidl’s Weight-Coded Approach

As already alluded to, out of the existing non-optimal algorithms Raidl’s weight-coded approach (1999) has been chosen for the analysis. For this purpose, we re-implemented the GA in Java. In the implementation we have followed as close as possible the description of the algorithm provided in Raidl’s work. The steady-state algorithm with population size 100 creates one child per generation, which replaces the worst individual in the population. The mutation rate is set to $3/n$, the recombination probability to 1, and the algorithm is stopped whenever 100000 evaluations are created without having found a new best solution. The selection procedure is a tournament selection with size two, and phenotypic duplicates are eliminated. All other operators are implemented as described in section 4.4.2.

All following examinations deal only with the weight-coded approach. Since the LB and MULTIPATH instances are all solved very fast to the optimum by CPLEX, these instances are not considered further. The focus lies solely on the MKNAPCB instances and their structural properties. Furthermore, the analysis of the weight-coded approach with all 270 instances is often too time-consuming. That is why we have chosen set 1, 5, and 9 (see table 5.3) whenever the number of runs must be restricted due to a lack of time. With this choice we assure that instances with different degrees of hardness are examined.
5.2 Performance

Among them, the easiest and the most difficult ones. Furthermore, two of the three sets are not solved by CPLEX in the experiments, so that it is necessary to investigate them with non-optimal algorithms.

In table 5.5 a comparison of the gaps presented in Raidl’s paper and the gaps of the re-implementation are compared (Raidl, 1999, p. 7). Furthermore, the gaps of the SS heuristic, which Raidl uses in his GA, are listed. All gaps are again in percent. For the re-implementation we computed gapB as additional information. All experiments with the weight-coded GA were made on a machine with 2×Dual Core AMP Opteron Processors 275 (4 cores, each 2.2GHz) with Linux (kernel 2.6.8), AMD 64 optimized, 4GB RAM. Since the running times of the GA and CPLEX are not compared, we decided to use this better computer to fasten the runs. The CPLEX runs could not be made on this machine, as it was not installed on it.

Table 5.5: Performance of Raidl’s weight-coded approach in comparison with the re-implementation. All gaps are in %. Each line represents 10 instances. For each instance 10 test runs were made.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>α</th>
<th>SS gapR</th>
<th>Ra(99) gapR</th>
<th>re-implementation gapR</th>
<th>σ</th>
<th>gapB</th>
<th>σ</th>
<th>time [sec.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100</td>
<td>.25</td>
<td>2.530</td>
<td>1.007</td>
<td>1.011</td>
<td>.0001</td>
<td>.023</td>
<td>.0001</td>
<td>33.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>1.621</td>
<td>0.453</td>
<td>0.465</td>
<td>.0001</td>
<td>.014</td>
<td>.0001</td>
<td>36.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>0.981</td>
<td>0.319</td>
<td>0.325</td>
<td>.0001</td>
<td>.007</td>
<td>.0001</td>
<td>35.98</td>
</tr>
<tr>
<td>5</td>
<td>250</td>
<td>.25</td>
<td>1.006</td>
<td>0.256</td>
<td>0.289</td>
<td>.0001</td>
<td>.690</td>
<td>.0001</td>
<td>129.58</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>0.491</td>
<td>0.127</td>
<td>0.135</td>
<td>.0000</td>
<td>.026</td>
<td>.0000</td>
<td>115.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>0.324</td>
<td>0.080</td>
<td>0.083</td>
<td>.0000</td>
<td>.007</td>
<td>.0000</td>
<td>89.16</td>
</tr>
<tr>
<td>5</td>
<td>500</td>
<td>.25</td>
<td>0.398</td>
<td>0.115</td>
<td>0.136</td>
<td>.0000</td>
<td>.065</td>
<td>.0000</td>
<td>307.89</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>0.219</td>
<td>0.053</td>
<td>0.059</td>
<td>.0000</td>
<td>.021</td>
<td>.0000</td>
<td>268.65</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>0.141</td>
<td>0.032</td>
<td>0.033</td>
<td>.0000</td>
<td>.009</td>
<td>.0000</td>
<td>222.85</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>.25</td>
<td>4.695</td>
<td>1.624</td>
<td>1.730</td>
<td>.0000</td>
<td>.171</td>
<td>.0000</td>
<td>39.64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>2.320</td>
<td>0.803</td>
<td>0.846</td>
<td>.0000</td>
<td>.055</td>
<td>.0000</td>
<td>39.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>1.618</td>
<td>0.493</td>
<td>0.502</td>
<td>.0000</td>
<td>.019</td>
<td>.0000</td>
<td>40.79</td>
</tr>
<tr>
<td>10</td>
<td>250</td>
<td>.25</td>
<td>1.998</td>
<td>0.589</td>
<td>0.653</td>
<td>.0000</td>
<td>-</td>
<td>-</td>
<td>132.87</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>0.924</td>
<td>0.276</td>
<td>0.294</td>
<td>.0001</td>
<td>-</td>
<td>-</td>
<td>125.33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>0.516</td>
<td>0.161</td>
<td>0.169</td>
<td>.0000</td>
<td>-</td>
<td>-</td>
<td>102.55</td>
</tr>
<tr>
<td>10</td>
<td>500</td>
<td>.25</td>
<td>1.006</td>
<td>0.332</td>
<td>0.354</td>
<td>.0000</td>
<td>-</td>
<td>-</td>
<td>282.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>0.408</td>
<td>0.150</td>
<td>0.156</td>
<td>.0000</td>
<td>-</td>
<td>-</td>
<td>291.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>0.290</td>
<td>0.085</td>
<td>0.089</td>
<td>.0000</td>
<td>-</td>
<td>-</td>
<td>241.28</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>.25</td>
<td>9.803</td>
<td>3.067</td>
<td>3.164</td>
<td>.0000</td>
<td>.286</td>
<td>.0000</td>
<td>51.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>4.453</td>
<td>1.376</td>
<td>1.411</td>
<td>.0000</td>
<td>.093</td>
<td>.0002</td>
<td>53.34</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>2.530</td>
<td>0.848</td>
<td>0.859</td>
<td>.0000</td>
<td>.035</td>
<td>.0001</td>
<td>61.84</td>
</tr>
<tr>
<td>30</td>
<td>250</td>
<td>.25</td>
<td>4.393</td>
<td>1.382</td>
<td>1.473</td>
<td>.0000</td>
<td>-</td>
<td>-</td>
<td>154.30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>1.719</td>
<td>0.609</td>
<td>0.617</td>
<td>.0000</td>
<td>-</td>
<td>-</td>
<td>157.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>1.269</td>
<td>0.348</td>
<td>0.345</td>
<td>.0000</td>
<td>-</td>
<td>-</td>
<td>140.09</td>
</tr>
</tbody>
</table>
5.3 Phenotypic Distance of Raidl’s Approach

Table 5.5: Performance of Raidl’s weight-coded approach in comparison with the re-implementation. (continued)

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>α</th>
<th>SS</th>
<th>Ra(99)</th>
<th>re-implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>gapR</td>
<td>gapR</td>
<td>gapB</td>
</tr>
<tr>
<td>30</td>
<td>500</td>
<td>.25</td>
<td>2.155</td>
<td>0.785</td>
<td>0.884</td>
</tr>
<tr>
<td>.50</td>
<td>.336</td>
<td>0.356</td>
<td>0.0001</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>.75</td>
<td>0.603</td>
<td>0.195</td>
<td>0.208</td>
<td>0.0000</td>
<td>-</td>
</tr>
<tr>
<td>average</td>
<td>1.864</td>
<td>0.589</td>
<td>0.617</td>
<td>0.0001</td>
<td>-</td>
</tr>
</tbody>
</table>

The gaps of the re-implementation are only slightly worse than the ones reported by Raidl. However, we are not aware of any deviance of the re-implementation to the original algorithm. Furthermore, one can see the excellent performance of the weight-coded GA. Its average gap is about three times better than the gap of the primal greedy heuristic. Moreover, the comparison of the SS gaps and Raidl’s gaps indicate the strong influence of the chosen bid-sorting heuristic on the meta-heuristic. The development of the two gaps along the increasing number of goods and bids is concurrent. This observation underlines the importance of the bid-sorting heuristic for the algorithm. Therefore, in future work, it is necessary to analyze the performance and functioning of bid-sorting heuristics in an even more enlarged scope than already done in the last section.

The running times show that the GA is quite fast. For the most simple instances the GA takes about 33 to 37 seconds, for the most difficult ones which are still solved optimally by CPLEX ($m = 30, n = 100$), the GA takes one minute. For the most complex instances the time is about 6 minutes. In CAs, it can be assumed that the participants will accept waiting times for the announcement of the allocation of several minutes. Wherefore the GA is an attractive algorithm for real-world applications.

5.3 Phenotypic Distance of Raidl’s Approach

We have just learned that Raidl’s algorithm works very well, but its performance is highly dependent on the used heuristic bias. This raises the question whether weight-coding really succeeds in exploring the search space or whether it concentrates only on exploiting regions near the solutions to the SS primal greedy heuristic. One approach to examine this aspect is to measure the distance between individuals. A high distance indicates a high degree of diversification, while a low distance signifies a concentration on only a small part of the search space. By observing distances throughout the whole search process, dynamics concerning exploration and exploitation can be analyzed.

In order to analyze distances, one has to decide whether to measure phenotypic or genotypic distances and which distance metric to choose. In section 4.2, it was already pointed out why in the case of MDPK and WDP the phenotypic search space should be focused in analyses. Furthermore, the Hamming distance was introduced as meaningful metric. With these tools, we can concentrate on the following questions. First, it is necessary to think about a reference point with which the measured distances can be compared. A
5.3 Phenotypic Distance of Raidl’s Approach

Hamming distance of 20 does not provide much information if it is unknown what the average Hamming distance between two individuals is. Thus, in the first part, the expected average distance for two random individuals will be computed, given a certain class of test instances. After the derivation of the formula, we will ask to what extent the solutions found by the GA differ from the solutions to the primal greedy heuristic used in the GA. As there are several parameters in Raidl’s approach which control the search process, it will be interesting if they affect the strength of the heuristic bias. The parameters observed are the perturbation size and the mutation rate. The third part addresses the dynamics of the distance throughout all computed generations. The average phenotypic distance of two individuals in the evolving populations will be compared with the reference point. A crucial aspect is to see how duplicate elimination affects the distances and what it contributes to the diversity of the population. In the last section newly created offspring will be examined. The distance to the current population reflects the degree of innovation which is implanted in the new generation.

5.3.1 Expected Distance of two Individuals

The average Hamming distance of two individuals depends on the length of the individual and the distribution of ones/zeros. Since the number of bids, the number of goods, and the tightness ratios of the test instances are given in advance, they can be used as parameters to determine the expected distance. Yet, the formula we develop in this section is only an approximation of the real average distance, as the tightness ratio does not determine exactly the number of ones. However, having, for example, an $\alpha$ of 0.50, 50% of the desired units are available. Therefore, about 50 percent of the bids can be accepted and set to one. Setting the maximum number of bits to one means creating only solutions in the boundary of the search space. This aspect is important, because all phenotypes of the GA also lie in the boundary.

In order to simplify the derivation of the formula, one of the two individuals is fixed. Only the second one varies in the positions of ones and zeros. The key idea for this simplification is that the individuals are drawn uniformly from all possible individuals. Therefore, having $k$ individuals and denoting the expected Hamming distance between two individuals $\text{ind}_i$ and $\text{ind}_j$ as $E[d(\text{ind}_i, \text{ind}_j)]$, it holds that

$$E_{i,j=1,...,k}[d(\text{ind}_i, \text{ind}_j)] = \sum_{i=1}^{k} p(\text{ind}_i) E_{j=1,...,k}[d(\text{ind}_i, \text{ind}_j)]$$

(5.10)

$$= E_{j=1,...,k}[d(\text{ind}_i, \text{ind}_j)]$$

since $E[d(\text{ind}_i, \text{ind}_j)] = E[d(\text{ind}_i, \text{ind}_j)]$ and $p(\text{ind}_i) = \frac{1}{k}$, $\forall i, j, l = 1, ..., k$.

As an example for the simplification consider individuals of length three with two ones, for example [110]. Computing the average distance of all possible individuals to this fixed individual is equal to computing the expected average distance of the second individual to any other individual with the same number of ones.

One might think that this simplification cannot be made, since the biased decoder works exactly opposed to that. It causes promising phenotypes to be over-represented and not uniformly distributed. Yet, the aim of this computation is not to determine the expected
5.3 Phenotypic Distance of Raidl’s Approach

Distance of two phenotypes in the GA, but to find a reference value indicating how strong
the heuristic bias is. Therefore, two uniformly created phenotypes must be compared with
the distances of phenotypes created by the GA.

In the remaining of this section, the formula is derived.

\[ n = \text{the vector representing an individual} \]
\[ p = \text{number of ones in a vector} \]
\[ q = \text{number of zeros in a vector} \]

so

\[ n = p + q \]
\[ ind_1 = \text{individual 1 (fixed)} \]
\[ ind_2 = \text{individual 2} \]

w.l.o.g. \( ind_1 := \underbrace{1\ldots1}_{p\text{-times}} \underbrace{0\ldots0}_{q\text{-times}} \)

**Lemma 1**

\[
\binom{n}{p} = \text{number of all possible individuals with } p \text{ ones and length } n \quad (5.11)
\]

**Proof:**

First take all possible \( n! \) permutations of the bit strings. The objective is to find all
possible positions that \( p \) ones can take. As the order of the ones is irrelevant (we do not
distinguish between the ones), we can divide the \( n! \) by the permutations of \( p \) ones, namely
\( p! \), the same holds for the number of zeros, \( n - p! \), leading to:

\[
\frac{n!}{p!(n-p)!} = \binom{n}{p} \quad (5.12)
\]

\( \square \)

**Theorem 2** With loss of generality assume that \( p \leq q \). If \( p > q \), just interchange the
two variables. The expected average distance between two random bit strings with length
\( n \), \( p \) ones, and \( q \) zeros is:

\[
E[d(ind_1, ind_2)] = \sum_{i=0}^{p} \frac{2i(p)^i(q)^i}{\binom{n}{p}} \quad (5.13)
\]

**Proof:**

The approach to prove this claim is the following. The variable \( i \) denotes the number of
positions where \( ind_1 \) has a one, but not \( ind_2 \). For each possible \( d(ind_1, ind_2) = 2i, i = 0, \ldots, p \), the number of possible \( ind_2 \) with distance \( 2i \) is counted. This number is divided by
the number of all possible individuals. This fraction represents the probability of having
two individuals with distance \( 2i \).
Lemma 1 proved the coefficient in the denominator of formula 5.13. What remains is the computation of the numerator, the number of \( ind_2 \) with distance 2 to \( ind_1 \). First, consider all possible \( ind_2 \) with distance 0 to \( ind_1 \). Obviously, there is only one possible combination, the one which is identical to \( ind_1 \). Now, we think of all \( ind_2 \) with distance \( 2 + 1 = 2 \). Note that, without loss of generality, the first \( p \) positions in \( ind_1 \) are ones and the rest zeros. A distance of two is reached when at one position out of the \( p \) positions there is a zero in \( ind_2 \). As the number of ones is assumed to be fixed with the given tightness ratio, the missing one in the first \( p \) positions of \( ind_2 \) must be allocated in the second half of the individual, where only zeros are located. How many possibilities are there, for allocating a zero in the first \( p \) positions and a one in the last \( q \) positions? Obviously, for each of the \( p \) positions, there are \( q \) possibilities to allocate the one, leading to a number of \( p \times q \) possibilities. Next, consider the case of a distance for \( i = 2 \), that is \( d(ind_1, ind_2) = 2 \times 2 = 4 \).

With the formula, the expected difference of two individuals is computed for some cases relevant for the analyses. As already pointed out, the tightness ratio is only an estimate for the number of ones in an individual. Therefore, we assume ranges in which the number of ones will probably be. If, for instance, the number of bids is 100 and the tightness ratio is 0.25, we assume a range of 20 to 30 ones. Since 25% of the capacity is available, it is very probable, that the allocations will accept between 20 and 30 bids. Note that this observation only holds for the MKNAPCB problems, where the expected number of units in each bid is the same, and also the expected number of units per good is identical. For tightness ratios of 50%, we assume a range from 45% and 55% to be set to one and for \( \alpha \) of 75%, between 70% and 80%. The resulting expected average differences for all cases are listed in table 5.6.

Table 5.6: Expected average distance between two uniformly drawn individuals for different parameter settings of the MKNAPCB instances.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \alpha )</th>
<th>accepted bids</th>
<th>exp. average distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.25</td>
<td>20-30</td>
<td>32</td>
</tr>
<tr>
<td>100</td>
<td>0.50</td>
<td>45-55</td>
<td>49.5</td>
</tr>
<tr>
<td>100</td>
<td>0.75</td>
<td>70-80</td>
<td>32</td>
</tr>
<tr>
<td>250</td>
<td>0.25</td>
<td>40-75</td>
<td>67.2</td>
</tr>
<tr>
<td>250</td>
<td>0.50</td>
<td>112.5-137.5</td>
<td>123.75</td>
</tr>
<tr>
<td>250</td>
<td>0.75</td>
<td>175-200</td>
<td>67.2</td>
</tr>
<tr>
<td>500</td>
<td>0.25</td>
<td>100-150</td>
<td>160</td>
</tr>
<tr>
<td>500</td>
<td>0.50</td>
<td>225-275</td>
<td>247.5</td>
</tr>
<tr>
<td>500</td>
<td>0.75</td>
<td>350-400</td>
<td>160</td>
</tr>
</tbody>
</table>
5.3 Phenotypic Distance of Raidl’s Approach

5.3.2 Distance to Primal Greedy Heuristics and CPLEX

In section 5.5 we alluded to the strong dependency between the performance of the weight-coded approach and the SS bid-sorting heuristic. This aspect is deepened in this section. To this end, we compute the Hamming distance between the best solution found by the reimplementation and the solutions of the most important bid-sorting heuristics. By testing different parameters for the perturbation size and the mutation rate, an insight into the influence of the heuristic bias is given. Furthermore, CPLEX is taken as reference, since it has no heuristic bias. The distances of the the primal greedy heuristics solutions among each other provide an idea to what extend their bias varies.

In table 5.7 distances of two test instances are demonstrated. We have decided to take one of the most easiest and one of the most complex instances CPLEX and the GA still solve optimally. Thereby, we can observe whether CPLEX and the GA find different optima in the search space. This would indicate that both algorithms focus their search on different regions. With these guidelines the two instances were drawn randomly from the data. The first instance has 5 goods, and 100 bids, and a tightness ratio of 0.25. The second one has, as well, 100 bids, but 10 goods and a tightness ratio of 0.75. The mutation rate of $3/n$ corresponds to the default rate. We augmented this rate to $10/n$ and $20/n$ to test the effect on the solutions. Additional experiments were made with a doubled perturbation size of 0.1. This size determines the degree of problem modification. In the last four columns the Hamming distances to the most important bid-sorting heuristics are presented. Distances are only computed between the optimal solutions of the respective algorithms.

Table 5.7: Hamming distance between solutions of bid-sorting heuristics, CPLEX, and the GA. MKNAPCB1_1 has 5 goods and 100 bids and MKNAPCB7_21 has 10 goods and 100 bids.

<table>
<thead>
<tr>
<th>algorithm</th>
<th>pert.</th>
<th>mut-rate</th>
<th>gapB[%]</th>
<th>NBP</th>
<th>SNBP</th>
<th>RLPS</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MKNAPCB1_1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBP</td>
<td></td>
<td></td>
<td>10.472</td>
<td>13</td>
<td>17</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>SNBP</td>
<td></td>
<td></td>
<td>1.551</td>
<td>13</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>RLPS</td>
<td></td>
<td></td>
<td>1.557</td>
<td>17</td>
<td>10</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td></td>
<td></td>
<td>7.707</td>
<td>17</td>
<td>10</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>CPLEX</td>
<td>0.05</td>
<td>3/100</td>
<td>0</td>
<td>17</td>
<td>12</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>GA</td>
<td>0.05</td>
<td>10/100</td>
<td>0.324</td>
<td>17</td>
<td>12</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>20/100</td>
<td>1.000</td>
<td>17</td>
<td>10</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>3/100</td>
<td>0.033</td>
<td>15</td>
<td>12</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>10/100</td>
<td>0.808</td>
<td>20</td>
<td>11</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>20/100</td>
<td>1.395</td>
<td>17</td>
<td>14</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td><strong>MKNAPCB7_21</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBP</td>
<td></td>
<td></td>
<td>1.867</td>
<td>8</td>
<td>12</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>SNBP</td>
<td></td>
<td></td>
<td>2.956</td>
<td>8</td>
<td>14</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>RLPS</td>
<td></td>
<td></td>
<td>0.667</td>
<td>12</td>
<td>14</td>
<td>-</td>
<td>7</td>
</tr>
</tbody>
</table>
Table 5.7: Hamming distance between solutions of bid-sorting heuristics, CPLEX, and the GA. (continued)

<table>
<thead>
<tr>
<th>algorithm</th>
<th>pert.</th>
<th>mut.rate</th>
<th>gapB[%]</th>
<th>NBP</th>
<th>SNBP</th>
<th>RLPS</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS</td>
<td></td>
<td></td>
<td>2.023</td>
<td>11</td>
<td>13</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>CPLEX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>23</td>
<td>21</td>
<td>24</td>
</tr>
<tr>
<td>GA</td>
<td>0.05</td>
<td>3/100</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>10/100</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>20/100</td>
<td>0.965</td>
<td>13</td>
<td>11</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>3/100</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>10/100</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>20/100</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>8</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 5.7 shows that the solutions of the primal greedy heuristics differ. However RLPS and SS bias in the same direction. In the experiments with the first instance, one can see that the optimal solution found by CPLEX is closest to the RLPS and the SS heuristic. The expected average distance in this case is 32 to 42. The sequence of the four distances 17, 12, 6, 6 reflects the performance of the heuristics. As shown in table 5.4, SS and RLPS perform best, followed by SNBP and NBP. The weight-coded approach finds exactly the same optimum as CPLEX when taken the default input parameters. The higher perturbation rate of 0.1 also leads to good results with the low mutation rate, but the optimum is not achieved. With augmented mutation rate, the gaps worsen. In the higher perturbed case, the distances also dissociate more and more from the SS solution. Concerning the second instance, which is lower restricted and includes more goods, three other aspects can be observed. First of all, CPLEX and the GA find an optimum in a very different region. With a distance of 11, the weight-coded algorithm is much closer to the SS heuristic than the CPLEX optimum with a distance of 24. This proves that in some cases there are several global optima. Second, the solution of the GA has a higher distance to the heuristic it is biased with than to the other ones. The final aspect we would like to mention is the robustness of the solutions for increasing perturbation size and mutation rate. Only in the case with a perturbation of 0.05 and a mutation rate of 20/100, the GA is unable to find the optimum.

The figures indicate that the GA reacts differently to an augmented modification of the original problem. In case with a low tightness ratio and a small number of goods, it behaves more sensitively to changes. We suppose that the higher degree of restrictions causes this behavior, since only a small change in the order of the bid list can have great impact on the acceptance of other bids, and with increasing perturbation of the problem, the probability of larger changes in the ordered list augments. This argument will be discussed further when examining the distances in the population. Furthermore, the table shows that the GA is rather strongly biased by the SS heuristic and exploits regions around its solutions. Nevertheless, in the second case, the GA even finds an optimum closer to solutions of other heuristics signifying a certain degree of exploration. Compared to the much higher distances of the CPLEX solution, however, we can conclude, that the weight-coded approach focuses more on exploiting than on exploring the search space. For the
highly restricted case with increased problem modification the search steps are apparently too large to exploit the region around the SS solution thoroughly.

To conclude, we can speculate that choosing the same perturbation size and mutation rate for all instances is inadequate. Depending on structural properties of the test data, the GA reacts differently on the same parameters. Furthermore, a more exploratory search seems not to improve the solutions. We get the impression that it is rather important to intensively exploit the search space around solutions to the primal greedy heuristic. Since the four presented bid-sorting heuristics differ, it might be better to exploit regions around all four solutions than to spend time in an extended exploration of the search space. Nevertheless, this analysis just provides first ideas about the search process. Further examination must be made to cement the propositions.

5.3.3 Distance during Search Process

In this part the distance between individuals is measured in two ways. One measures the average distance between all individuals per generation. By means of the distance inherent in a whole population one can examine if it develops more diversely or specifically. Beginning with the initial population, a comparison between the expected average distance computed in section 5.3.1 and the distance between individuals in the population of the weight-coded approach provides valuable insights. If in the initial population the distance is much lower than the expected average distance, the modified problems are either all very similar to each other or the heuristic bias is very strong. The second approach is to measure the distance of newly created offspring to the current population. This indicates to which extent the offspring is able to contribute to the evolvement of the whole population.

Distance of all Individuals in Population

In order to investigate the first aspect, phenotypical distances in runs of three test instances are observed. The three instances are chosen so that each number of goods and bids and each tightness ratio are observed. Instance 5-11 and 9-21 are not solved optimally by CPLEX. In every 100th generation, the average distance of two individuals in the current population is determined. Furthermore, a comparison between runs with duplicate elimination and without duplicate elimination is made to measure their contribution to diversity.

In figure 5.2 the results are presented. In the first picture, a MKNAPCB test instance with 5 goods, 100 bids, and $\alpha = 0.25$ is displayed. On the x-axis, the generations are shown and on the y-axis the average distance of two individuals in the current population is marked. The dashed lined curve illustrates the development of the distances without duplicate elimination, the solid line with duplicate elimination. The setup of the other two figures corresponds. Illustration (b) shows MKNAPCB5-11 with 10 goods, 250 bids, and $\alpha = 0.50$, while figure (c) presents an instance with 30 goods, 500 bids, and a tightness ratio of 0.75.

The distances do not differ much in the initial population, although the initial population can have duplicates in the no duplicate elimination case. In the three cases the values are 2.7, 7.8, and 12. Compared to the expected average distances the values are very low. In all instances it is only about 7%-8% of the expectation for random individuals in the
5.3 Phenotypic Distance of Raidl’s Approach

Figure 5.2: Distances of two individuals in populations throughout the whole search process. Comparison between runs with and without duplicate elimination.

(a) Instance with 5 goods, 100 bids, and a tightness ratio of 0.25.

(b) Instance with 10 goods, 250 bids, and a tightness ratio of 0.50.

(c) Instance with 30 goods, 500 bids, and a tightness ratio of 0.75.

boundary of the search space. This signifies that the perturbation of the weights is not very strong or the heuristic bias is so intense that it maps very different weights to similar solutions. Yet, as all presented primal greedy heuristics map surjectively to the boundary of the search space, all solutions should be possible with the right choice of weights.

Most obvious in all figures is the different development of the two curves. In case duplicates are not eliminated, the population converges early as the distance drops to values around zero. Nevertheless, also duplicate elimination does not prevent the population from fast convergence. The population reaches as well a point of stagnation were all individuals are extremely similar. In other words, duplicate elimination can only prevent the population of being totally uniform. Only the smallest degree of diversity, which is the smallest average distance two individuals can have, is guaranteed. Furthermore, the convergence speed seems to be correlated with the tightness ratio. Although in figure (c) many more decision variables exist, the convergence is faster. Further investigations concerning this observation follow in the next sections.

Finally, it is crucial to analyze if the extreme exploitative search for the most part of the runs still improves the results. If so, the early convergence needs not necessarily to be a
disadvantage of the algorithm. The number of evaluations after which no better solution was found as well as the performance of duplicate elimination (DE) and no duplicate elimination (NDE) are provided in table 5.8.

<table>
<thead>
<tr>
<th>test data</th>
<th>gap[%]</th>
<th>number evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DE</td>
<td>NDE</td>
</tr>
<tr>
<td>MKNAPCB-1-01</td>
<td>0.833</td>
<td>0.989</td>
</tr>
<tr>
<td>MKNAPCB-5-11</td>
<td>0.311</td>
<td>0.290</td>
</tr>
<tr>
<td>MKNAPCB-9-21</td>
<td>0.182</td>
<td>0.238</td>
</tr>
</tbody>
</table>

In two of three cases the runs with DE perform better. In the second case, the results are close, but NDE is more successful. Nevertheless, in average, a design without duplicates performs better. Additionally, one can note that the runs with NDE all find their optimum earlier than in runs with DE. This observation verifies our statement that duplicate elimination guarantees a minimal degree of diversification, and in most cases, this diversity succeeds in finding better solutions also late in the search process. Finally, a very important aspect can be inferred from the table. Although in figures 5.2 it can be seen that in all cases the algorithm converges before the 20000 generation, the algorithms still finds better solutions long after this point of time. This observation indicates again the success of a rather exploitative search in the neighborhood of the primal greedy heuristics. It is in line with the results gained from further analyses.

**Distance of Children to Population**

In this part, the minimal distance between a newly created child and the existing population is examined. As minimum we define:

\[
d_{\text{min}}[\text{pop, child}] = \min_{k=1, ..., \text{popSize}} \{d[\text{ind}_k, \text{child}], \text{ind}_k \in \text{pop}\}.
\] (5.14)

In our view, taking the minimum measures best the degree of innovation the child incorporates into the generation. Taking, for example, the average or the maximum distance would not inform about already existing individuals which are almost identical to the new offspring.

In figures 5.3 \(d_{\text{min}}\) is plotted. In figure (a) a test instance with 100 bids, 5 goods, and \(\alpha = 0.25\) is shown, in figure (b) one with 250 bids, 10 goods, and \(\alpha = 0.50\), and in figure (c) and (d) one with 500 bids, 30 goods, and \(\alpha = 0.75\). On the x-axis, the distance is marked and on the y-axis the frequency with which this distance appears in the observed period. Two periods are chosen, the first 3000 generations and the last 3000 generations of a run. In pictures (a), (b), and (c) four curves are drawn, two for the two periods with duplicated elimination (DE) and two for the two periods without duplicate elimination (NDE). For example in the first test set, during the last 3000 generation without duplicate elimination between 800 and 900 children are created with minimal distance 0 to the
current population. In other words, almost every third to fourth child has distance 0. Figure (d) depicts the $d_{\text{min}}$ for every 100th generation.

In figures (a), (b), and (c) several observations are made. First, the curves of runs with and without duplicates do not differ significantly. This fact supports the point that reproduction creates rather dissociated children when most parents have at least a minimal distance of one or two. For the case that there are many duplicates in the population, as it can be seen for the curves $\text{NDE: last 3000 gen.}$, a lot of children with minimal distance zero are created. In the smallest instance ca. 850/3000 children have distance zero in the middle sized one there are ca. 340/3000 and (c) 320/3000. These high ratios are again an indicator for the stagnation of the algorithm without duplicate elimination.

Second, comparing all three figures, the curves of the last generations dissociate more and more from the curves showing the first generations. The fraction $\text{distance of child/expected average distance}$ is about the same for the three instances in the first 3000 generations. For MKNAPCB-1-01 it is about $7/37 \approx 0.2$ and for the other two cases 0.14. Yet, for the last generations, the absolute minimal distance of the child is almost equal for all cases, being about four to seven in average, although the expected average distance is much
5.3 Phenotypic Distance of Raidl’s Approach

higher in the case with a high number of bids, see table 5.6. Therefore, MKNAPCB-9-21
concentrates on a very small part of the search space at the end of the search, given its
huge size.

In figure (d) we see that the distance of new offspring diminishes with an increasing
number of generations. It oscillates mostly in the range between 4 to 9. The decrease
is caused by the decreasing distance in the population. The parents become similar and
will therefore create offspring with a low distance. The expected average distance of two
random individuals for this test instance is between 160 and 210 and therefore much higher.
The corresponding plots for figures (a) and (b) are displayed in the appendix, figure A.1.
There, the distance of the offspring varies mainly in the range from 3 to 12. Given that the
plot measures a minimal distance and given that the average distance in the population
is between two and three (see figure 5.2), one can state that the reproduction creates
offspring with a high distance.

In summary, in figures 5.2 and 5.3 we have noticed that the distance between individuals
in the population is much smaller than the distance of newly created children. One could
say that the algorithm focusses on a small search space and tries to explore by creating
offspring which is quite different from the current population. The question is if the newly
created offspring really has a chance to stay in the population. Since the steady-state
algorithm replaces always the worst child, a newly created child which explores the search
space and has to traverse a region with low fitness will be replaced immediately in the
next generation. Figure 5.2 has indicated that the diversity in the population diminishes
dramatically in the first generations, consequently, the offspring seems not to succeed in
implanting enduring new material in the population.

To examine the quality and the distance of the offspring further, another experiment
is conducted. For all MKNAPCB instances the average \( d_{\text{min}} \) of the created offspring is
compared with the \( d_{\text{min}} \) of the best child. We suppose that a rather exploitative search
is more successful than an exploratory one, and believe that the distance of the best
individual created will be smaller than the average distance of children. The results of the
experiments are presented in table 5.9.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>( \alpha )</th>
<th>( d_{\text{min}} ) average</th>
<th>( d_{\text{min}} ) best individual</th>
</tr>
</thead>
<tbody>
<tr>
<td>MKNAPCB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>0.25</td>
<td>6.0</td>
<td>3.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>4.9</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>3.8</td>
<td>3.2</td>
</tr>
<tr>
<td>5</td>
<td>250</td>
<td>0.25</td>
<td>7.2</td>
<td>4.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>6.1</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>4.7</td>
<td>4.3</td>
</tr>
<tr>
<td>5</td>
<td>500</td>
<td>0.25</td>
<td>9.0</td>
<td>5.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>6.8</td>
<td>4.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>5.5</td>
<td>5.0</td>
</tr>
</tbody>
</table>
Table 5.9: Distance of best individual created compared with average distance of created children. (continued)

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>α</th>
<th>(d_{\text{min}}) average</th>
<th>(d_{\text{min}}) best individual</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100</td>
<td>0.25</td>
<td>6.6</td>
<td>3.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>5.3</td>
<td>2.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>4.0</td>
<td>3.1</td>
</tr>
<tr>
<td>10</td>
<td>250</td>
<td>0.25</td>
<td>7.7</td>
<td>4.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>6.2</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>4.8</td>
<td>3.8</td>
</tr>
<tr>
<td>10</td>
<td>500</td>
<td>0.25</td>
<td>8.6</td>
<td>3.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>6.9</td>
<td>4.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>5.4</td>
<td>4.8</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.25</td>
<td>6.7</td>
<td>3.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>5.6</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>4.2</td>
<td>4.0</td>
</tr>
<tr>
<td>30</td>
<td>250</td>
<td>0.25</td>
<td>7.3</td>
<td>3.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>6.0</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>4.8</td>
<td>4.1</td>
</tr>
<tr>
<td>30</td>
<td>500</td>
<td>0.25</td>
<td>8.0</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>6.4</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>5.1</td>
<td>3.9</td>
</tr>
<tr>
<td>average</td>
<td></td>
<td></td>
<td>6.1</td>
<td>4.0</td>
</tr>
</tbody>
</table>

The table verifies the suggestion. The best individual is found in the neighborhood of the current population. The average \(d_{\text{min}}\) is in average 1.53 times larger than the \(d_{\text{min}}\) of the best individual.

Another aspect revealed by the table is the influence of the tightness ratio on the average \(d_{\text{min}}\). However, there seems to be no coherence to the \(d_{\text{min}}\) of the best offspring whose values are quite constant. Since the average value is not sufficient for a detailed analysis, the observation is deepened in figure 5.4. Here, the \(d_{\text{min}}\) is plotted for different tightness ratios. The first three pictures present the minimal distance of children for different tightness ratios in the first 3000 generations. The last picture, (d), also shows the last 3000 generations for the most difficult test instance. The x-axis and the y-axis are equivalently labelled as the ones in figure 5.3. The frequencies for different distances are reproduced.

In all figures showing the first generations, the same tendency can be observed. With an increasing tightness ratio, the distance diminishes. The less constraint the problem is, the less diverse gets the population. This effect can mainly be caused by two reasons. Either in the phenotypes it must be more probable to have a small distance when there are more bids accepted, or the heuristic bias is stronger in case of a higher tightness ratio. To discuss these ideas, let us consider once more the formula for the expected average distance. In figure 5.5 the expected average distance for all possible number of accepted bids in the case with 100 bids is plotted. It shows that the maximum distance should be achieved when half of the bids are accepted. This contradicts to the assumption that the
5.3 Phenotypic Distance of Raidl’s Approach

distance gets smaller with increasing number of accepted bids. Thus, the first explanation does not apply, therefore the heuristic bias must be stronger with increasing tightness ratio. A similar conclusion was already drawn in section 5.3.2 when comparing the distance of the GA to the primal greedy heuristics for varied perturbation sizes. Nevertheless, we would like to explain this effect more detailed.

A stronger heuristic bias might seem astonishing, since neither the prices, nor the weights change with increasing tightness ratio. Therefore, the ordering of the weight vector is independent from $\alpha$. We assume the reason for a strong heuristic bias to be the following. Small changes in the order of the weight vector lead to fewer phenotypic changes when there is a weakly restricted problem. That is why with the same strength of problem modification the decoder maps to a smaller region of the search space when the instance is weakly restricted. For example, with a tightness ratio of 75% in average only $\frac{1}{n+0.75}$ of available units are consumed by a bid, while in case of 25%, $\frac{1}{n+0.25}$ are consumed. Therefore, a modification of the ordered bid list has much more effect with scarce resources. Neighbored genotypes will be mapped to a more diverse space of phenotypes than in a weakly restricted problem. The consequence to draw from this conclusion is the same as

Figure 5.4: Minimal distance of children to current population. In figures (a), (b), and (c) the first and in figure (d) the last 3000 generations are plotted for different tightness ratios, namely 0.25, 0.5, and 0.75.

(a) First 3000 generations of test instance with 5 goods and 100 bids.
(b) First 3000 generations of test instance with 10 goods and 250 bids.
(c) First 3000 generations of test instance with 30 goods and 500 bids.
(d) Last 3000 generations of test instance with 30 goods and 500 bids.
5.4 Phenotypic Duplicates of Raidl’s Approach

In this section, the number of phenotypic duplicates produced by the weight-coded approach will be analyzed. The number of duplicates provides information about two aspects we are interested in. First, it is again a measurement for the intensity of exploration and exploitation. If the number of duplicates is large, the search process probably concentrates on a small region; it is more exploitative. If the number is small, the algorithm is either very exploratory or it succeeds in searching a small region without creating too many duplicates. Second, the number of duplicates indicates whether the algorithm is inefficient. Producing a lot of duplicates when duplicate elimination is used just yields an overhead and makes the algorithm slower.

All previous results showed that the GA searches in a rather small region of the search space, the one which is close to the solution of the bid-sorting heuristic. Nevertheless, we have learned that this high heuristic bias works well for the MDKP and the WDP. The search space is extremely large and the mapping to feasible solutions very redundant, so that this approach might be the best to tackle the difficult search process. Because of the exploitative search, we expect a high number of duplicates. For making the algorithm more efficient, we hope to be able to reduce this number by adopting the perturbation size and the mutation rate. We are convinced that these changes must be made dependent on structural properties of the test instances. Still, we have to guarantee that with the higher perturbation of the problem, the solution quality does not suffer.

In table 5.10 the produced phenotypic duplicates for different perturbation sizes are listed. The two perturbation sizes chosen are larger than the default value of 0.05, since we expected that a higher perturbation rate would produce less duplicates. Experiments with other sizes showed the same behavior. For each of the three perturbation sizes tested, the ratio between duplicates and the total number of produced offspring is shown in percent (dup.). A duplicate is produced if the new child already exists in the current population. The method for measuring the number of duplicates is similar to the one used in literature (Raidl and Gottlieb, 2005, Gottlieb, 1999a, p. 58). Since 10 runs for
each of the 10 different instances per row were realized, also the standard deviation ($\sigma$) is presented. Furthermore, the average gap for all runs is displayed in percent. As in this section only different parameter settings are compared to each other, the gapR is sufficient as comparison criterion.

Table 5.10: Produced duplicates to the current population. For each of the three perturbation sizes tested, the duplicate ratio $\%$, the standard deviation of this ratio, and the gapR $\%$ are provided.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>$\alpha$</th>
<th>pert. 0.05</th>
<th></th>
<th>pert. 0.1</th>
<th></th>
<th>pert. 0.3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>dup.</td>
<td>$\sigma$</td>
<td>gapR</td>
<td>dup.</td>
<td>$\sigma$</td>
<td>gapR</td>
<td>dup.</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>-------</td>
<td>---------</td>
<td>------</td>
<td>-------</td>
<td>---------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>.25</td>
<td>1.67</td>
<td>0.0007</td>
<td>1.011</td>
<td>1.48</td>
<td>0.0003</td>
<td>1.008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>2.74</td>
<td>0.0008</td>
<td>0.465</td>
<td>2.63</td>
<td>0.0011</td>
<td>0.468</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>6.19</td>
<td>0.0015</td>
<td>0.325</td>
<td>5.60</td>
<td>0.0019</td>
<td>0.320</td>
</tr>
<tr>
<td>5</td>
<td>250</td>
<td>.25</td>
<td>0.75</td>
<td>0.0006</td>
<td>0.289</td>
<td>0.88</td>
<td>0.0005</td>
<td>0.317</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>1.27</td>
<td>0.0008</td>
<td>0.135</td>
<td>1.50</td>
<td>0.0010</td>
<td>0.150</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>2.96</td>
<td>0.0015</td>
<td>0.083</td>
<td>3.37</td>
<td>0.0019</td>
<td>0.091</td>
</tr>
<tr>
<td>5</td>
<td>500</td>
<td>.25</td>
<td>0.48</td>
<td>0.0004</td>
<td>0.136</td>
<td>0.60</td>
<td>0.0006</td>
<td>0.195</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>1.13</td>
<td>0.0009</td>
<td>0.059</td>
<td>1.53</td>
<td>0.001</td>
<td>0.097</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>2.06</td>
<td>0.0014</td>
<td>0.033</td>
<td>2.61</td>
<td>0.0018</td>
<td>0.045</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>.25</td>
<td>1.35</td>
<td>0.0006</td>
<td>1.730</td>
<td>1.22</td>
<td>0.0007</td>
<td>1.752</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>2.58</td>
<td>0.0014</td>
<td>0.846</td>
<td>2.48</td>
<td>0.0011</td>
<td>0.861</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>5.80</td>
<td>0.0020</td>
<td>0.502</td>
<td>5.28</td>
<td>0.0016</td>
<td>0.496</td>
</tr>
<tr>
<td>10</td>
<td>250</td>
<td>.25</td>
<td>0.72</td>
<td>0.0003</td>
<td>0.653</td>
<td>0.89</td>
<td>0.0005</td>
<td>0.722</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>1.56</td>
<td>0.0015</td>
<td>0.294</td>
<td>1.89</td>
<td>0.0008</td>
<td>0.340</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>3.25</td>
<td>0.0016</td>
<td>0.169</td>
<td>3.30</td>
<td>0.0035</td>
<td>0.180</td>
</tr>
<tr>
<td>10</td>
<td>500</td>
<td>.25</td>
<td>0.62</td>
<td>0.0006</td>
<td>0.354</td>
<td>0.83</td>
<td>0.0009</td>
<td>0.489</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>1.24</td>
<td>0.0012</td>
<td>0.156</td>
<td>1.60</td>
<td>0.0014</td>
<td>0.215</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>2.73</td>
<td>0.0031</td>
<td>0.089</td>
<td>3.24</td>
<td>0.0020</td>
<td>0.116</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>.25</td>
<td>1.41</td>
<td>0.0007</td>
<td>3.164</td>
<td>1.39</td>
<td>0.0008</td>
<td>3.219</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>2.71</td>
<td>0.0013</td>
<td>1.411</td>
<td>2.64</td>
<td>0.0013</td>
<td>1.415</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>6.36</td>
<td>0.0030</td>
<td>0.859</td>
<td>6.15</td>
<td>0.0046</td>
<td>0.867</td>
</tr>
<tr>
<td>30</td>
<td>250</td>
<td>.25</td>
<td>1.42</td>
<td>0.0014</td>
<td>1.473</td>
<td>1.36</td>
<td>0.0010</td>
<td>1.610</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>2.52</td>
<td>0.0021</td>
<td>0.617</td>
<td>2.56</td>
<td>0.0019</td>
<td>0.679</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>4.39</td>
<td>0.0037</td>
<td>0.345</td>
<td>4.75</td>
<td>0.0048</td>
<td>0.367</td>
</tr>
<tr>
<td>30</td>
<td>500</td>
<td>.25</td>
<td>1.40</td>
<td>0.0014</td>
<td>0.884</td>
<td>1.38</td>
<td>0.0012</td>
<td>1.097</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>2.58</td>
<td>0.0014</td>
<td>0.356</td>
<td>2.72</td>
<td>0.0022</td>
<td>0.448</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.75</td>
<td>4.53</td>
<td>0.0040</td>
<td>0.208</td>
<td>4.92</td>
<td>0.0041</td>
<td>0.254</td>
</tr>
</tbody>
</table>

The table demonstrates that the gap for the higher perturbation sizes increases. Apparently, the standard parameters for the Raidl’s approach are already well optimized. Only in three cases the gap improves. Concerning instances with 5 goods and 100 bids, the gap for a perturbation size of 0.1 is with 1.008% and 0.320% slightly better than the one
achieved by the default setting. The same holds for the instances with 10 goods, 100 bids, and \( \alpha = 0.75 \). Nevertheless, these improvements are so small, that a change of the perturbation size seems unnecessary. Concerning the duplicate ratio, we see that it improves for a perturbation of 0.1 in all cases of 100 bids, and for the instances with 30 goods, 250 and 500 bids, and \( \alpha = 0.25 \). In all other cases, it augments with increasing perturbation. Once again, Raidl’s approach seems to already minimize the number of duplicates by choosing the right perturbation size. The improvement in the cases with 100 bids can be attributed to the fact that it is more probable to produce a duplicate in a string with 100 bits than in the strings with 250 and 500 bits.

There is the trend that with increasing number of bids, the number of duplicates decreases. Consequently, in cases with only few bids, the perturbation size must be chosen rather large and for instances with many bids, rather small. Furthermore, there is a trend to specialization with increasing tightness ratio. It goes along the trend of decreasing distances with increasing \( \alpha \). The decoding procedure maps the genotypes to a smaller space of phenotypes in the weakly restricted cases. The reason for this has already been explained in the last section where we have suggested increasing the perturbation size. Unfortunately, the table shows that this suggestion has not the desired effect, since the gaps do not improve significantly.

Overall, the number of produced duplicates is quite low. A much higher number was expected, because of the highly exploitative search observed during the analysis of distances. The reason for this might be the following. The algorithm, indeed, searches in a small part of the search space, but in the population only 100 individuals are stored at one point of time which makes it unlikely to produce a duplicate. Furthermore, if duplicates are eliminated, parents cannot be identical, therefore, it is unlikely that a duplicate of the parents is produced. Additionally, in one of the previous analyses, we have observed, that children have a rather high distance to their parents. We stated that, most probable, new offspring is supposed to be replaced rather soon in the next generations. This makes it improbable that new offspring is equal to existing individuals. Nevertheless, it might be the case that the population often produces identical children. To verify the suggestion, another kind of measuring duplicates is tested. Not only are the number of duplicates to the current population counted, but the number of duplicates in relation to all produced duplicates throughout the search process. That is, during the run all individuals ever generated are stored and whenever a new child is produced it is tested for equality to all previous produced individuals. This duplicate ratio is denoted with total duplicate ratio, while the former one will from now on be called direct duplicate ratio. The total duplicate ratio provides valuable information, for example about an inefficient oscillating search between local optima. If the search process only jumps back and forth between the same regions, the direct duplicate ratio might be very low, while the total duplicate ratio could be very high. Furthermore the direct duplicate ratio depends more on parameters like population size and replacement scheme. If, for example, new offspring only replaces the worst individual if its fitness is better, the population could produce always the same child without making any progress. In this case, the direct duplicate ratio would be zero while the total duplicate ratio would uncover this inefficiency. On the other hand, a small population fosters small direct duplicate ratios while the size of the population has less influence on the number of totally produced duplicates.
5.4 Phenotypic Duplicates of Raidl’s Approach

Table 5.11: In total produced duplicates for three different sets of MKNAPCB test instances in %.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>α</th>
<th>pert. 0.05</th>
<th>pert. 0.1</th>
<th>pert. 0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>tot.dup.</td>
<td>mut.dup.</td>
<td>tot.dup.</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>0.25</td>
<td>10.91</td>
<td>11.35</td>
<td>9.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>21.86</td>
<td>23.47</td>
<td>19.40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>49.32</td>
<td>47.93</td>
<td>41.11</td>
</tr>
<tr>
<td>10</td>
<td>250</td>
<td>0.25</td>
<td>4.32</td>
<td>11.09</td>
<td>4.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>11.55</td>
<td>22.40</td>
<td>13.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>28.57</td>
<td>46.82</td>
<td>28.84</td>
</tr>
<tr>
<td>30</td>
<td>500</td>
<td>0.25</td>
<td>6.09</td>
<td>10.62</td>
<td>5.61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>14.14</td>
<td>22.03</td>
<td>13.95</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>29.34</td>
<td>46.86</td>
<td>29.54</td>
</tr>
</tbody>
</table>

The results for the total duplicate ratio for three sets of test instances are presented in table 5.11. Again, set 1, 5, and 9 were chosen. As additional information we examine how often the mutation operator causes no changes in the children (mut.dup). The results verify the conjecture. The total number of duplicates (tot.dup.) is much higher than the number of direct produced duplicates. Particularly the amount of duplicates for high tightness ratios is alarming. The rates are around 40% to 50% indicating a high inefficiency in the search process. Therefore, we strongly advice that in future analyses one should also focus on the total duplicate ratio.

Another aspect to examine is the very high percentage of produced duplicates by the mutation operator. In cases of $\alpha = 0.75$, 50% of the mutation operations are unnecessary. This shows that the mutation rate is much too low for the weakly restricted instances. Latter aspect is now analyzed by varying the mutation rate. Maybe this can help to reduce the number of duplicates. In table 5.12 we compare the direct duplicate ratio, the total duplicate ratio, and the gapR of runs with different mutation rates. For each of the ten available test instances per row, ten runs were made so that each entry in the table averages 100 values.

Obviously both the direct duplicate ratio and the total duplicate ratio diminish drastically when the mutation rate is doubled or tripled. However, in most cases, the gapR does not decrease. Only in the cases with high tightness ratio, the gaps stay almost constant while the efficiency can be improved. The effect of the tightness ratio reflects the results from former analyses. For weakly restricted instances, the heuristic bias is stronger, therefore, the problem modification should be increased to diversify the search process. Apparently, not an increase of the perturbation size, but an increase of the mutation rate accomplishes the desired behavior. More experiments should be conducted with a more sensitive adaptation of the mutation rate to test whether not only the efficiency, but also the solution-quality can be improved further.
Table 5.12: Direct duplicates and total duplicates for different mutation rates.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>α</th>
<th>dir.dupl.[%]</th>
<th>tot.dupl.[%]</th>
<th>gapR[%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100</td>
<td>0.25</td>
<td>1.41</td>
<td>0.22</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>2.29</td>
<td>0.75</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>5.55</td>
<td>3.21</td>
<td>1.55</td>
</tr>
<tr>
<td>250</td>
<td>0.25</td>
<td>0.62</td>
<td>0.06</td>
<td>0.01</td>
<td>4.23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>1.09</td>
<td>0.31</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>2.74</td>
<td>1.52</td>
<td>0.72</td>
</tr>
<tr>
<td>500</td>
<td>0.25</td>
<td>0.38</td>
<td>0.02</td>
<td>0.00</td>
<td>2.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>0.95</td>
<td>0.25</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>1.85</td>
<td>1.09</td>
<td>0.46</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>0.25</td>
<td>1.06</td>
<td>0.13</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>2.22</td>
<td>0.70</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>5.17</td>
<td>2.89</td>
<td>1.30</td>
</tr>
<tr>
<td>250</td>
<td>0.25</td>
<td>0.67</td>
<td>0.06</td>
<td>0.01</td>
<td>3.80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>1.33</td>
<td>0.39</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>2.80</td>
<td>1.52</td>
<td>0.78</td>
</tr>
<tr>
<td>500</td>
<td>0.25</td>
<td>0.54</td>
<td>0.03</td>
<td>0.00</td>
<td>2.94</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>1.06</td>
<td>0.29</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>2.30</td>
<td>1.39</td>
<td>0.60</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.25</td>
<td>1.19</td>
<td>0.42</td>
<td>2.18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>2.27</td>
<td>0.62</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>5.77</td>
<td>2.96</td>
<td>1.32</td>
</tr>
<tr>
<td>250</td>
<td>0.25</td>
<td>1.18</td>
<td>0.12</td>
<td>0.01</td>
<td>5.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>2.20</td>
<td>0.59</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>4.20</td>
<td>2.19</td>
<td>0.97</td>
</tr>
<tr>
<td>500</td>
<td>0.25</td>
<td>1.09</td>
<td>0.11</td>
<td>0.00</td>
<td>4.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>2.15</td>
<td>0.65</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>4.28</td>
<td>2.34</td>
<td>0.83</td>
</tr>
</tbody>
</table>

5.5 Improving the State of the Art

With the gained knowledge about structural properties of test instances and different algorithms, a profound appraisal of potential improvements is possible. The analysis of test instances and the performance of existing solutions revealed that optimal algorithms, in particular CPLEX, perform excellent on instances with sparse constraint matrices. Once the matrices become denser, in the experiments CPLEX fails to solve instances with more than 10 goods and 250 bids. Since the problem is \(NP\)-complete, it seems adequate to give up optimality for the latter instances and to focus on non-optimal algorithms. Consequently, we have chosen to examine one of the best meta-heuristics further, Raidl’s
weight coded approach. The central point of this GA is the applied heuristic bias, the SS bid-sorting heuristic. Therefore, the performance of different bid-sorting heuristics was observed. The good results indicated that the application of such sorting-heuristics is very reasonable. Nevertheless, the heuristics perform differently well dependent on structural properties of the test instances.

The further analyses concerning distances between individuals and produced duplicates showed that the search process is extremely exploitative due to the strong heuristic bias. The distance in the population drops to a minimal level early in the search process. Although children have a rather high distance to the existing population, they do not diversify the search. Probably, they are replaced rather soon. Further investigation should be made to verify the latter suggestion. We have learned that the best child found is always in the very neighborhood of the current population. Hence, searching by only small search steps seems to be very successful. Another important fact is that both the heuristic bias and the performance increase when the instance is weakly restricted. This confirms that a strong heuristic bias affects the search positively.

As consequence of the very exploitative search, the number of duplicates is quite high. Yet, this effect can only be observed when counting the total number of duplicates. In order to improve the inefficiency caused by too many duplicates, the perturbation rate and the mutation rate were augmented. The adaptation of the perturbation size had mostly negative effects, on the gap as well as on the duplicate ratio. Therefore, an adaptation of the perturbation size will not lead to large improvements of the algorithm. Changing the mutation rate helped a lot to diminish the number of duplicates, but, as well, the gap did not improve significantly. Nevertheless, we could show that the mutation in the default settings of Raidl’s algorithm is almost non-existing, since the mutation operator often lets the individuals unchanged. We think that a more sensitive adaptation of the mutation rate will help the algorithm to decrease the number of duplicates and to improve its performance.

All aspects of the analyses point at one main idea for improvement. The GA should keep its exploitative search behavior, but should restart with different primal greedy heuristics. The reason for this kind of change in the algorithm should be clear directly from the results of the analyses. The strong heuristic bias is very successful since near the primal greedy solutions good solutions can be found. Furthermore, the presented bid-sorting heuristics bias the problem to different regions. Consequently, the GA should exploit the regions close to the bid-sorting heuristics thoroughly, and explore the search space by examining regions around solutions to various bid-sorting heuristics.

Table 5.13: The performance of the GA with different heuristic biases and different perturbation sizes in %.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>α</th>
<th>SS gapR pert.0.05</th>
<th>SNBP gapR pert.0.06</th>
<th>NBP gapR pert.0.08</th>
<th>NBP gapR pert.0.1</th>
<th>NBP gapR pert.0.5</th>
<th>min. gapR</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100</td>
<td>.25</td>
<td>1.010</td>
<td>1.004</td>
<td>1.004</td>
<td>1.049</td>
<td>1.021</td>
<td>0.998</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.50</td>
<td>0.465</td>
<td>0.454</td>
<td>0.455</td>
<td>0.501</td>
<td>0.453</td>
<td>0.451</td>
</tr>
</tbody>
</table>
The following experiments give an idea how these changes can improve the performance of the weight-coded approach. Table 5.13 compares the gaps achieved by the GA for different heuristic biases. The default sorting-heuristic Raidl used is the SS heuristic with perturbation size 0.05. We tested two of the discussed alternatives, the SNBP and the NBP. An application of the RLP primal greedy heuristic in the weight-coded approach turned out to be not practicable with the current implementation. Since the RLP solution assigns a lot of zeros to the bids, there are not enough gradations among the bids which makes a meaningful sorting difficult.

This section is supposed to only provide an idea of what improvements are possible. Therefore, no in-depth parameter tuning was done for SNBP and NBP. Some experiments indicated that perturbation sizes of 0.06 and 0.08 work well with the GA using SNBP, and sizes of 0.1 and 0.5 for the case when NBP was integrated in the GA. Furthermore, no experiments were made with an adaptation of the mutation rate. Although, in our view, an adaptation of the mutation rate in dependence on the tightness ratio might improve the performance of the algorithm further. Once more, only the reduced set of test instances was chosen for the experiments. Each line summarizes ten instances with only one run per instance. In case of the default GA with SS heuristic, the average value from the runs before is taken. In the last column (min.), for each instance the minimal gapR of the three heuristics is chosen.

The results show that the instances with 100 bids and 5 goods, and the ones with 10 goods, 250 bids, and \( \alpha = 0.25 \) are best solved by the weight-coded approach using SNBP as heuristic bias. For the other instances and in average over all instances, the default approach suggested by Raidl is the best one. However, restarting the runs with the proposed different heuristic bias and storing always the best solution, yields a significant improvement of the gap, as can be seen in the last column. The question remains whether restarts lead to a tremendous increase in running time. Therefore, we compare the number of evaluations until the optimum is reached.
5.5 Improving the State of the Art

Table 5.14: Number of evaluations for different heuristic biases.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>α</th>
<th>SS eval. pert.0.05</th>
<th>SNBP eval. pert.0.06</th>
<th>NBP eval. pert.0.08</th>
<th>NBP eval. pert.0.1</th>
<th>NBP eval. pert.0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100</td>
<td>0.25</td>
<td>37024</td>
<td>22270.4</td>
<td>28630.3</td>
<td>12268.8</td>
<td>5554.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>25250.8</td>
<td>22839.4</td>
<td>19198.9</td>
<td>21926.7</td>
<td>8086.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>2387.3</td>
<td>18953.6</td>
<td>17441.9</td>
<td>20786.5</td>
<td>16661.1</td>
</tr>
<tr>
<td>10</td>
<td>250</td>
<td>0.25</td>
<td>90211.9</td>
<td>57281.2</td>
<td>64761.1</td>
<td>53841.9</td>
<td>66728.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>71020.7</td>
<td>74691.1</td>
<td>70495.5</td>
<td>42807.6</td>
<td>47550.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>62052.8</td>
<td>41945</td>
<td>68572.4</td>
<td>41195.8</td>
<td>39190.7</td>
</tr>
<tr>
<td>30</td>
<td>500</td>
<td>0.25</td>
<td>81770.8</td>
<td>64240.7</td>
<td>83140.5</td>
<td>57224.8</td>
<td>60335.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>88551.3</td>
<td>63912.5</td>
<td>85622.7</td>
<td>64830</td>
<td>63387</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>40453.2</td>
<td>64409.1</td>
<td>69472.4</td>
<td>60307.1</td>
<td>78927.8</td>
</tr>
<tr>
<td>average</td>
<td></td>
<td></td>
<td>55413.64</td>
<td>47838.11</td>
<td>56370.63</td>
<td>41687.69</td>
<td>42935.72</td>
</tr>
</tbody>
</table>

Table 5.14 reveals that the number of evaluations for SNBP and NBP is smaller than the number for the default setting. Furthermore the stopping criteria of Raidl to halt the run after 100000 evaluations without improvement seems rather generous. Changing the stopping criteria so that the algorithm stops earlier and restarts with another bid-sorting heuristic is more appropriate. This change should lead to an improvement of the algorithm. Finally, we think that more improvements can be made by adopting the mutation rate slightly in dependence on the tightness ratio. However, further experiments have to be conducted to confirm these suggestions.
6 Summary, Conclusions, and Further Work

The purpose of this diploma thesis is the analysis and comparison of various algorithms for the multi-unit WDP and the MDKP in order to provide a unified view of both research domains. Furthermore, the detailed examination of a weight-coded GA is supposed to indicate to further improvements of existing meta-heuristics. This chapter recapitulates the content of this work and points out its contributions.

6.1 Summary

In chapter 2, an introduction to the fundamentals of combinatorial auctions was provided. First, we summarized the aspects which motivate to deal with this subject. The main point is that in combinatorial auctions agents can express synergies between the goods they want to obtain, which diminishes the risk of receiving unfavorable combinations of goods. The covering of this risk leads to a win-win situation: the agents can bid higher prices, and the auctioneer can make higher revenue. Afterwards, some basic concepts and results in the field of mechanism design were regarded. We have learned that the VCG mechanism circumvents the negative results of the Gibbard-Satterthwaite theorem by assuming the agents to have quasi-linear preferences. Due to the utilization of a special payment scheme, the VCG mechanism reveals to be allocative-efficient and strategy-proof. With this background about mechanism design, we were able to focus on one of its implementations, namely auctions and, in particular, CAs. We presented potential and realized applications of CAs to further motivate their importance. Having laid the basics for the problem domain of CAs, we finally specified one of its major problems, the allocation problem or WDP. Other problems of computational complexity in CAs, such as valuation complexity, strategic complexity, and communication complexity, were not considered in this work. Unfortunately, the \( \text{NP} \)-completeness of the WDP usually makes it impossible to apply the favorable VCG mechanism. That is because the VCG mechanism requires the WDP to be solved optimally. Consequently, only for small input sizes, which can still be solved to optimality, truthful bidding of the agents can be assumed. Finally, we showed that the IP formulation of the multi-unit WDP is equivalent to one variant of KPs, the MDKP. Since KPs are often dealt with in abstraction of specific applications, an introduction, comparably detailed as the one to combinatorial auctions, was unnecessary.

In order to enable a profound analysis of different algorithms for the WDP and MDKP domains, a literature overview of existing solutions was presented in chapter 3. We showed that optimal algorithms in both domains use similar concepts; they mostly apply branch and bound methods. For the general multi-unit case, however, much fewer algorithms have been proposed than for the specialized single-unit case. Furthermore, non-optimal algorithms, and in particular meta-heuristics, have been focussed on by research on the MDKP, while being neglected by research on the WDP. Since meta-heuristics like GAs have demonstrated their success in the field of MDKP, we claimed for concentrating on them aiming at an improved version applicable to both domains.
Therefore, in chapter 4, we examined GAs in detail. First, their functionality was described emphasizing the importance to design them in such way that the trade-off between exploration and exploitation is addressed adequately. On the one hand, the operators must guarantee that promising regions of the search space are exploited thoroughly, and, on the other hand, a certain degree of diversification should assure an exploration of the search space. Since the incorporation of problem-specific knowledge in the design of a GA usually improves its performance, we scrutinized the structure of the search space and possible heuristic biases to guide the search. A basic point to consider in the search space of the WDP and the MDKP is the handling of constraints. We alluded to the fact that the optimal solution for the IP must be located in the boundary of the feasible search space. The feasible search space contains all solutions which do not violate any constraint. Furthermore, since many solutions in the search space are infeasible, often a mapping from the infeasible solutions to the feasible ones is applied. As the feasible region of the search space is, particularly in highly restricted problems, smaller than the infeasible region, a highly redundant representation is likely.

To guide the search process towards promising regions within the boundary of the search space, different bid-sorting heuristics from the literature were introduced. The heuristics can be applied as stand-alone primal greedy heuristic, or they can be included in a meta-heuristic as problem-specific bias. We showed that the complexity of the sorting-heuristics varies a lot and that some of them even require a relaxed version of the original LP to be solved. The relaxation is achieved by allowing the binary decision variables of the original problem to be continuous in the range from 0 to 1.

In the last section of this chapter, we applied the knowledge about the search space and heuristic biases to the presentation and valuation of existing GAs for the MDKP. According to Gottlieb (1999a), we separated the GAs into three classes: (1) direct representation in the complete search space, (2) direct representation in the feasible search space, and (3) indirect representation in the feasible search space. The GAs belonging to the first class usually apply penalty functions to guide the search towards the feasible regions. Oftentimes, these approaches waste resources by searching predominantly in the infeasible search space. Moreover, their results are worse than the ones reached by other representations. In the second class, repairing methods and local optimization are used to guarantee the producing of high-quality offspring in the feasible region. Here, a very good algorithm has been introduced by Chu and Beasley (1998). Its success can be attributed to the strong heuristic bias. In the third class, various indirect representations were presented. Thereby, we indicated Raidl’s steady state weight-coded approach to be one of the most successful GAs for solving the MDKP (Raidl, 1999). As genotype Raidl chooses a string of real valued weights, each weight is assigned to one bid. The mapping from genotypes to phenotypes consists of two steps. First, the price of each bid is modified with its corresponding weight, and then a primal greedy heuristic maps this modified problem to a phenotype by deciding which bid to include in the allocation. Since we suspected this algorithm to remain improvable due to its inefficiency caused by the highly-redundant representation of real values, we decided in favor of its further examination in the subsequent chapter. Furthermore, Raidl’s and Chu and Beasley’s approach use the same bid-sorting heuristic. Thus, insights gained from Raidl’s GA could also help to improve Chu and Beasley’s algorithm.
6.1 Summary

In chapter 5, existing algorithms were analyzed. A basic concern was a detailed examination of test instances. First, because of the high dependency between structural properties of test instances and the performance of algorithms, and, second, since in the field of combinatorial auctions, real-world data is rare and meaningful artificial test instances must be generated. Since research in CAs has already addressed the topic of structural properties of test instances, we presented an extraction of the proposed attributes (Leyton-Brown et al., 2006). Research in MDKP has mostly used the tightness ratio to characterize an instance. We differentiated between properties describing the strength of restriction, such as the tightness ratio, the BG edge density, and the BG clustering coefficient, and, on the other hand, the density of the constraint matrix, reflected by the BGG good degree of the good nodes. We supposed that a constraint matrix with very few zero-entries makes the problem more complex, and is therefore harder to solve for most algorithms. The direction in which the strength of restriction would affect the hardness of problems was unclear at this stage of research. The three chosen types of test instances, LB, MULTIPATH, and MKNAPCB, were analyzed afterwards according to their structural properties. Concerning the LB instances generated, we approximated the BG edge density by computing its expectation. This result and the low tightness ratio indicate the high restrictiveness of this type of problem instances. However, the constraint matrix have turned out to be very sparse. The next data generation method was a multi-unit version of the CATS PATH distribution, denoted as MULTIPATH. The tightness ratio points to a slightly less restricted problem than the LB instances, and the constraint matrix is determined to be quite sparse. Concerning the last type of instances, taken from the MDKP literature, the structural properties differ a lot from the previous ones. The high tightness ratio, the very low edge density, and the resulting low clustering coefficient hint at a weakly restricted problem, while the dense constraint matrix points at high complexity.

In the next part of the analyses, the performance of four approaches was tested: CAMUS, a specified optimal branch and bound algorithm for multi-unit WDPs, the commercial solver ILOG CPLEX 9.0, several primal greedy heuristics, and Raidl’s weight-coded GA. The performance was judged with respect to structural properties of the test data explained in the first analysis. Although CAMUS is very fast when computing the smallest LB test instances, for all other, more complex instances, it is rather slow or fails. It collapses when the constraint matrix is dense, or the number of goods and bids is large. CPLEX performs very well on all instances taken from the CA literature. However, once the number of goods and bids of the MKNAPCB instances increases, they become too complex so that in the experiments CPLEX either requires long time or cannot solve them at all. We argue that the structure of these hard MKNAPCB test instances is also relevant for scenarios in combinatorial auctions, such as network bandwidth allocation problems. That is why, research on combinatorial auctions should consider non-optimal algorithms developed in research on MDKP. Due to CPLEX’ failure on complex instances, we concentrated on non-optimal algorithms in the remaining of the chapter. The experiments with primal greedy heuristics showed that they are very simple, but perform well. Consequently, using them as heuristic bias in meta-heuristics seems quite appropriate. Furthermore, their performance is highly dependent on structural properties of the test instances, such as the number of goods, bids and the tightness ratio. We suggested different explanations for these effects and explained why the performance decreases for more
restricted problems. Another interesting observation was the excellent performance of the heuristics on instances in which prices were generated totally randomly, that means independently from the items bids included. This fact indicates that the method for generating prices in artificial test instances has a major influence on performance. We argued that as soon as the generation of prices favors certain structures in bids, all heuristics favoring the same structures will perform well, wherefore, research should bear in mind how prices of artificial test instances are generated when testing algorithms. The final experiments on performance was conducted with a re-implementation of Raidl’s weight-coded approach. The results depict its good performance and fast running times which are much better than the ones by CPLEX. In average the GA achieves a gap to the optimum smaller than 0.6% and computes the most difficult instances in 364 seconds. However, the performance of the GA is strongly influenced by the performance of the SS heuristic used to bias the search.

The next two sections concentrated on examining the search process of Raidl’s GA in detail. The aim was to analyze which characteristics of the search are successful and which might be further improved. Thereby, the Hamming distance between the phenotypes was chosen as one method to measure the degree of exploration and exploitation of the search process. We examined the heuristic bias which might hinder the population from staying diverse, as well as the convergence behavior of the algorithm. Furthermore, the contribution phenotypic duplicate elimination yields to diversity was inspected. First, we have derived a formula to compute the expected average distance of two random individuals in the boundary of the search space. The expected average distance served as a reference point for the subsequent observations. Since we aimed to measure the strength of the heuristic bias, the distance between the solutions of the GA and the solutions of the primal greedy heuristics was computed afterwards. The analysis has shown that the GA is strongly biased towards the solution of the applied SS bid-sorting heuristic. Furthermore, we observed that CPLEX finds optima located in regions quite distant from the ones explored by the GA. The solutions of the bid-sorting heuristics, NBP, SNBP, and SS differ moderately, while the solutions of the RLPS heuristic are quite close to the SS heuristic. Additionally, variations of the perturbation size and the mutation rate indicate a sensitive reaction of the GA when computing highly restricted test instances. We have suggested that when adopting parameters, the intensification of variation must be adjusted to the degree of restriction. The succeeding analyses dealt with the distances measured during the search process. First, we addressed the average distance between individuals in the population and then the minimal distance of new offspring to the current population. The results allude to a very exploitative search process. Both for the runs without duplicate elimination and the runs with duplicate elimination after no more than 20000 generations, the average distance is close the minimal distance. Nevertheless, further experiments have shown that the best solution can still be found after this point of convergence. Consequently, the extensive exploitative search seems to be necessary for finding the best solutions. This conclusion has been supported by analyses of the minimal distance of newly created offspring to the current population. Although the offspring created has a rather moderate distance to the population, much higher than the distance within the population, apparently, the children do not succeed in diversifying the search. Other experiments have demonstrated that the best individuals are always found in the neighborhood of the current
population with a distance lower than the average distance of new children. This effect is much stronger for low tightness ratios. Overall, the average minimal distance diminishes when the strength of restriction decreases. We assert this to be due to a more intense heuristic bias for instances with a high tightness ratio.

The last sequence of analyses dealt with the number of produced duplicates. The influence of parameters on this number was examined in order to answer the question, if the algorithm could improve its efficiency by diminishing the amount of duplicates, while not worsening the solution quality. The direct duplicate ratio has been defined in accordance to the literature as the number of produced duplicates to the current population divided by the total number of new offspring. First, experiments with an augmented perturbation size were conducted. However, neither the number of duplicates reduce significantly, nor does the quality of solutions improve. We argue that the direct duplicate ratio does not measure the number of duplicates adequately. A better notion would be to compare a new child to all produced individuals throughout the search process, denoted by the total duplicate ratio. A GA which, for instances, jumps back and forth between the same two local optima might produce no direct duplicates indicating an efficient search. In reality, however, this behavior is inefficient since during the whole search process identical offspring is often created. Furthermore, we have explained that the total duplicate ratio is more independent of parameters like the replacement scheme and the population size. The following experiments with the total duplicate ratio uncovered a high inefficiency in the search process. Up to 50% of the offspring is a duplicate. Further analyses have demonstrated that the mutation is mainly responsible for this high rate. In half of the cases with high tightness ratio, the mutation does not change the individual at all. Consequently, last analyses were done with a varied mutation rate. An increase of the mutation rate diminishes the number of duplicates drastically, but in most cases, the solution-quality cannot be improved when duplicating or triplicating the rate. Nevertheless, for the instances with high tightness ratio the gap can be kept constant while improving the efficiency of the search.

6.2 Conclusions

From this work basically four conclusions concerning the multi-unit WDP and the MDKP can be drawn.

**Research on both domains can enrich each other.** The goal was to compare existing algorithms from the WDP and the MDKP in detail, aiming at a more intense understanding and a mutual inspiration of both research areas. The comparison has shown that, on the one hand, similar ideas for algorithms have been developed, but, on the other hand, the focus of research has differed. Research on the WDP has concentrated more on optimal algorithms by restricting the problem, or by solving only easy test instances. Contrary to this, research on the MDKP has also focused on non-optimal algorithms, in particular meta-heuristics, for solving more complex test instances. We think that the sophisticated and specialized algorithms designed in the domain of WDP can contribute to the field of MDKP as well. Other applications of the MDKP might learn, for example, from the ideas of pre-processing and bounding applied in the WDP branch and
bound algorithms. Furthermore, due to the increasing interest in e-auctions, enlarged and sophisticated real-world applications are the scenarios of the future. This development demands for algorithms which can deal with more complex data. Therefore, research in CAs should clearly focus on non-optimal algorithms which can guarantee fast response times and high-quality solutions. Hereby, it could directly make use of algorithms like the weight-coded GA developed for solving the MDKP.

**Artificial test instances must be generated with care.** In the analyses we have listed different structural properties of test instances and showed how instances can be described by those properties. The degree of restriction, the number of goods and bids, and the density of the constraint matrix had a major influence on the performance of algorithms. Therefore, methods for generating artificial test instances should be designed with care, and statements about the performance of algorithms should be judged prudently, having structural properties of test instance in mind. Concerning real-world application, we advise to analyze the data before applying an algorithm blindly. Possible approaches to figure out characteristics of existing instances were described in this work. Knowing how algorithms react to different properties, might help to make a good choice in advance, particularly with regard to the limits of optimal algorithms.

**Meta-heuristics are successful when incorporating a strong heuristic bias.** The analyses made in this work demonstrated that meta-heuristics for the MDKP and the multi-unit WDP are rather successful when exploiting a small region of the search space thoroughly than exploring larger regions. To guarantee a search in promising regions of the search space, a strong heuristic bias is an effective mean. Both direct and indirect representations take advantage of this approach. Concerning direct representations a biased mapping from the infeasible search space to the feasible search space improves the search, and referring to indirect representation the mapping from genotype to phenotype is positively influenced by a strong heuristic bias. Nevertheless, a very local search bears the risk of inefficiency, since usually many duplicate solutions are produced. This work uncovered such inefficiencies by introducing a new duplicate ratio: the number of total duplicates. We concluded that, in case of the weight-coded approach, an adaptation of the mutation rate improves the solution-quality while producing less duplicates. However, more experiments must be carried out with a more sensitive adaptation of the mutation rate. Increased perturbation sizes seem not to enhance the performance which might be caused by the choice of the LOGNORM distribution of weights.

**The weight-coded approach can improve by applying different bid-sorting heuristics.** The main conclusion of this work is that the weight-coded GA should keep to its exploitative search behavior and could slightly improve its inefficiency by increasing the mutation rate for some instances. Nevertheless, much better results can be made when restarting the algorithm with different heuristic biases. We showed that an exploitative search nearby solutions of various bid-sorting heuristics is the right approach for balancing out the trade-off between exploration and exploitation.
6.3 Further Work

This work exemplifies that the awareness of equivalent problems crossing different research domains can help a lot to improve research. In future work more effort should be laid on seeking out similar or even equivalent problems addressed in other domains to fasten and improve research. When, for example, thinking about algorithms for CAs with non-free disposal, which means the auctioneer has to allocate all offered goods, research should first examine solutions found for the set packing problem, due to its equivalence to the this kind of auction.

Test instances for the multi-unit CA are relatively rare. Therefore, future research should think more about potential applications and meaningful generation methods for generating artificial test instances. Further examinations about structural properties of test instances will help to do so. Since in this work all properties for the WDP were taken from the literature on the single-unit case, it is necessary to investigate more detailed properties of the multi-unit case. Research similar to the analysis of Leyton-Brown et al. (2006) could be done for finding properties making the multi-unit case harder.

Concerning the application of algorithms in real-world scenarios, we believe that non-optimal algorithms such as Raidl’s weight-coded approach (1999) are appropriate methods to apply. For the instances tested, they provided solutions very close to the optimum and were much faster than optimal algorithms. However, guidelines should be formulated describing under which condition optimal algorithms can still be applied and when auctioneers should make use of non-optimal approaches.

An interesting approach is combining the weight-coded approach or other meta-heuristics with branch and bound methods. Gallardo et al. (2005) have recently hybridized a general branch and bound method with the GA by Chu and Beasley (1998) to solve the MDKP. They achieved very good results. We think that hybridizing more specialized branch and bound methods, like a multi-unit version of CABOB, with meta-heuristics might improve the results further (Sandholm et al., 2005). On the one hand, the very good solutions found by a meta-heuristic early on in the process can fasten the pruning of sub-trees. On the other hand, in this work it was found out, that, for instance, the weight-coded approach is successful when exploiting regions around good solutions. The branch and bound method could communicate such promising regions to the meta-heuristic.

Furthermore we think that applying tabu search might be a very promising approach for the WDP and the MDKP. A tabu search could exploit the search space nearby solutions of the primal greedy heuristics while avoiding the production of duplicates. Consequently, we expect tabu search to provide the same solution quality as GAs by being more efficient. More literature research on already existing tabu search algorithms for the two problem domains is necessary to consider this idea further.

Relating to genetic algorithms and, particularly, to the weight-coded approach, we will do further research on various bid-sorting heuristics. New heuristics must be found, and more studies must be conducted to learn about the influence of structural properties of test instances on the bid-sorting heuristics. This knowledge might even help to discover new heuristics with high-quality solutions. Furthermore, appropriate re-starting procedures must be found so that increasing running times can be prevented. Additionally, it will be interesting to test the performance of Chu and Beasley’s algorithm (1998) with other
bid-sorting heuristics. Concerning a possible self-adaptation of the perturbation size and mutation rate in the weight-coded approach, we believe that further experiments must be carried out with other perturbation methods, such as UNI, RELUNI and LOG. Specifically the additive composition of the weight and the price in the UNI and RELUNI method will react differently on a variation of the perturbation size, possibly leading to an improved duplicate ratio. Furthermore, experiments in Raidl’s work showed that the NORMLOG method takes more evaluation than the other methods. Therefore, a closer examination of those methods might be of great interest. Finally, other recombination methods creating children more similar to the parents might improve the algorithm, since this work has shown that the best individuals found are always in the neighborhood of the current solutions, while newly created offspring with the current method does not succeed in diversifying the population.
## Appendix

Table A.1: Optima for the MKNAPCB test instances not listed in the OR-library.

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>new optima</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>500</td>
</tr>
<tr>
<td>01:120148</td>
<td>02:117879</td>
<td>03:121131</td>
</tr>
<tr>
<td>06:122024</td>
<td>07:119127</td>
<td>09:121586</td>
</tr>
<tr>
<td>11:218428</td>
<td>12:221202</td>
<td>13:217542</td>
</tr>
<tr>
<td>15:218966</td>
<td>16:220530</td>
<td>17:219989</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>17:43574</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>100</td>
</tr>
<tr>
<td>05:021844</td>
<td>09:022525</td>
<td>13:041630</td>
</tr>
</tbody>
</table>
Figure A.1: Minimal distance of children to the current population with duplicate elimination. Every 100th generation is measured.
Bibliography


Eidesstattliche Erklärung

Ich versichere, dass ich meine Diplomarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mannheim, den 26.10.2006

Jella Pfeiffer