Master’s degree thesis

LOG950 Logistics

Heuristics for Binary Integer Programming Problems

Alexandr Reznik

Number of pages including this page: 62

Molde, 28.05.2021

Molde University College
Specialized University in Logistics

Mandatory statement

Each student is responsible for complying with rules and regulations that relate to examinations and to academic work in general. The purpose of the mandatory statement is to make students aware of their responsibility and the consequences of cheating. Failure to complete the statement does not excuse students from their responsibility.

*Please complete the mandatory statement by placing a mark **in each box** for statements 1-6 below.*		
1.	**I/we hereby declare that my/our paper/assignment is my/our own work, and that I/we have not used other sources or received other help than mentioned in the paper/assignment.**	☒
2.	**I/we hereby declare that this paper** 1. Has not been used in any other exam at another department/university/university college 2. Is not referring to the work of others without acknowledgement 3. Is not referring to my/our previous work without acknowledgement 4. Has acknowledged all sources of literature in the text and in the list of references 5. Is not a copy, duplicate or transcript of other work	Mark each box: 1. ☒ 2. ☒ 3. ☒ 4. ☒ 5. ☒
3.	I am/we are aware **that any breach of the above will be considered as cheating, and may result in annulment of the examination and exclusion from all universities and university colleges in Norway for up to one year, according to the Act relating to Norwegian Universities and University Colleges, section 4-7 and 4-8 and Examination regulations** section 14 and 15.	☒
4.	**I am/we are aware that all papers/assignments may be checked for plagiarism by a software assisted plagiarism check**	☒
5.	**I am/we are aware that Molde University College will handle all cases of suspected cheating according to prevailing guidelines.**	☒
6.	**I/we are aware of the University College's rules and regulation for using sources**	☒

Personal protection

Personal Data Act

Research projects that processes personal data according to Personal Data Act, should be notified to Data Protection Services (NSD) for consideration.

Have the research project been considered by NSD? ☐yes ☒no

- If yes:

Reference number:

- If no:

I/we hereby declare that the thesis does not contain personal data according to Personal Data Act.: ☒

Act on Medical and Health Research

If the research project is effected by the regulations decided in Act on Medical and Health Research (the Health Research Act), it must be approved in advance by the Regional Committee for Medical and Health Research Ethic (REK) in your region.

Has the research project been considered by REK? ☐yes ☒no

- If yes:

Reference number:

Publication agreement

ECTS credits: 30

Supervisor: Lars Magnus Hvattum

Agreement on electronic publication of master thesis

Author(s) have copyright to the thesis, including the exclusive right to publish the document (The Copyright Act §2).

All theses fulfilling the requirements will be registered and published in Brage HiM, with the approval of the author(s).

Theses with a confidentiality agreement will not be published.

I/we hereby give Molde University College the right to, free of charge, make the thesis available for electronic publication: ☒yes ☐no

Is there an agreement of confidentiality? ☐yes ☒no

(A supplementary confidentiality agreement must be filled in)

- If yes:

Can the thesis be online published when the period of confidentiality is expired? ☐yes ☐no

Date: 28.05.2021

Preface

This thesis was written according to the requirements for the Master of Science in Logistics degree. The thesis was written at Molde University College – Specialized University in Logistics. The work was supervised by Professor of Quantitative Logistics of Molde University College (Norway) Lars Magnus Hvattum.

I would like to thank Lars Magnus Hvattum for the motivation, inspiration, supprot provided during the work on this thesis, for the valuable input and for the amazing jokes that made the work on the thesis even more enjoyable. I would also like to thank to Håkon Bentsen and Lars Magnus Hvattum for providing the code base for the implementation of the algorithm.

Summary

This thesis focuses on creating a construction heuristic algorithm for the general binary integer problem. A greedy construction heuristic is created and different components are tested in order to obtain a good algorithm. Greedy Randomized Adaptive Search Procedure (GRASP) based on the greedy construction is implemented and tested. Conclusions regarding the possibility of using GRASP for solving binary integer problem are made. The way of combining the algorithms implemented during the work on the thesis with an improvement heuristic in order to get better results is shown.

The result of the thesis can be used during the further research of heuristic approaches for solving binary integer problem.

Contents

List of Tables

List of Figures

1.0 Introduction

Many optimization problems with real world applications can be modeled as binary optimization problems (BIP). Some applications of binary integer problems in logistics are airline crew scheduling, facility location problems, cutting stock problems. In addition, many different planning problems can be formulated as binary optimization problems.

Crew scheduling problem is solved to assign crew in order to operate transportation systems. Every huge company has to solve this problem in order to maintain the transportation system working. All the airline companies are trying to assign crew in an optimal way to reduce the cost of operating and to avoid delays and cancelations of flights.

Facility location problem is also very important for logistic companies. The optimal placement of the facilities helps to reduce transportation costs while having dangerous materials far from housings.

Cutting stock problem is very important for the paper industry (Kallrath et al. 2014). The decisions made by the stakeholders influence both the company's revenue and the global climate change. Paper industry influences the deforestation and therefore it has an influence on global warming. A more efficient use of paper will help to reduce this influence.

The assembly line balancing problem is widely used to schedule manufacturing. It allows to distribute task required to manufacture product among several workstations taking into account the precedence relations between the tasks. Mixed-model assembly line balancing problem was formulated as binary integer problem by Gökċen & Erel (1998).

With the growth of huge companies and the world in general the amount of the available information increases. In order to get higher profit companies try to solve different problems with better efficiency. This results in more constraints taken into account when solving binary integer problems.

The resulting binary integer problems are usually containing a huge number of constraints and very hard to solve. The exact methods use unacceptably long time to solve these problems. That is why the reasonable choice would be to use metaheuristic algorithms.

The lack of solvers for this specific problem opens a space for research. The direction of the research is focused on the creating a heuristic algorithm for the variety of binary integer problems. The existing solvers usually focus on solving specific binary integer problem (e.g. multidimensional knapsack problem (Cappanera and Trubian 2005)) or they are targeting a more wide class of problems such as mixed integer programming.

The goal of the research is to examine whether using Greedy Randomized Adaptive Search Procedure (GRASP) is a reasonable strategy for solving BIP. The research focuses on selecting strategies for different parts of the algorithm suitable for solving BIP. Another question of the research is whether GRASP can be combined with other metaheuristic algorithms, namely improvement metaheuristic, in order to improve their results and performance.

2.0 Literature Review

The existing solvers usually focus on solving specific binary integer problem (e.g. multidimensional knapsack problem (Cappanera and Trubian 2005)) or they are targeting a more wide class of problems such as mixed integer programming (Cplex 2009; Benoist et al. 2011).

2.1 Solving general BIP

There are several papers describing methods for the general BIP.

There are several contributions for solving BIP using heuristic methods. All the contributions are using improvement algorithms and this leaves space for a research of a construction algorithm for BIP.

Bertsimas, Iancu, and Katz (2013) created a pseudo-polynomial local search algorithm for BIP. The test results are presented for set covering and set packing problems.

Gortazar et al. (2010) present a black box scatter search for general BIP tested on different classes.

The idea of solving optimally a linear programming problem first letting the variables to be in range $[0, 1]$ and then moving to a BIP solution using a heuristic was described by Balas and Martin (1980) and recently used by Al-Shihabi (2021). The last paper focuses more on multidemand multidimensional knapsack problem rather than on general BIP.

An approach for sloving optimum satisfiability problem can be applied to general BIP (Jeong and Somenzi 1993). The algorithm presented is based on Binary Decision Diagrams. Authors claim that any BIP can be converted to an optimum satisfiability problem. However, the resulting problem can be too large and this can make the method impractical.

There are also methods based on exhaustive search and branch-and-bound strategies (Marinescu and Dechter 2010)(Baessler 1992). The study of Balas (1965) presenting an additive algorithm is extended by Glover (1965) and later by Geoffrion (1967).

2.2 Greedy Randomized Adaptive Search Procedure and construction heuristics

Greedy Randomized Adaptive Search Procedure (GRASP)(Feo and Resende 1989) is a metaheuristic algorithm for constructing solutions that was succesfully applied to a number of problems.

Different components of GRASP and successful implementation techniques with parameter tuning approaches are described by Resende and Ribeiro (2003). GRASP was applied to a boolean optimization problem, namely to the Maximum Satisfiability problem (Resende, Pitsoulis, and Pardalos 1997) (Resende and Feo 1996) and later to Weighted Maximum Satisfiability problem resulting in the solutions better than the ones obtained by commercial solvers (Hvattum, Løkketangen, and Glover 2005).

Festa and Resenda published several papers with the annotated bibliography of GRASP (Festa and Resende 2002; 2009b; 2009a).

Another problem that can be formulated as BIP where GRASP was applied is two-partition problem (Laguna, Feo, and Elrod 1994).

While some papers describe the use of GRASP for constructing solution (Vianna and Arroyo 2004) there are a number of papers describing improvement metaheuristics or population-based metaheuristics starting from random solution without using any construction heuristic other than random (Hristakeva and Shrestha 2004),(Lai, Hao, and Yue 2019). But there are works (Duarte and Martí 2007) using a combination of GRASP and an improvement metaheuristic to achieve good results.

3.0 Problem description

Binary integer problem can be formulated as:

$$\max Z = \sum_{j=1}^{n} c_j x_j \,, \tag{3.1}$$

subject to

$$b_i^l \le \sum_{j=1}^{n} a_{ij} x_j \le b_i^u, \qquad i = 1, \dots, m, \tag{3.2}$$

$$x_j \in \{0,1\}, \qquad j = 1, \dots, n, \tag{3.3}$$

where all the coefficients a_{ij}, b_i, c_j are integers (Bentsen and Hvattum 2020).

Equality constraints can be represented as two inequality constraints with different signs $\le$ and $\ge$. $\ge$-constraints can be transformed into $\le$-constraints by multiplying by -1 but this is not used to leave more information about the problem structure. All the non-integer rational coefficients in the constraints can be transformed into integers by multiplying each constraint by the least common multiply of the denominators of all coefficients. The non-integer rational coefficients in the objective function can be avoided by using the same approach.

The solution for the problem is a vector $x = (x_1, x_2, \dots, x_n) \in \{0,1\}^n$.

The following terms are introduced for the description of different methods:

A set of unassigned variables $V^{\#}$ is a set containing all the variables that haven't been assigned a value during the construction of the solution.

Activity level is the value of the left-hand side of the constraint. It is introduced as

$$AL_i = \sum_{j=1}^{n} a_{ij} x_j \tag{3.4}$$

Let minimum activity level of a constraint i be:

$$MinAL_i = \min_{x \in \{0,1\}^n} \left(\sum_{j=1}^{n} a_{ij} x_j \right) \tag{3.5}$$

Let maximum activity level of a constraint i be:

$$MaxAL_i = \max_{x \in \{0,1\}^n} \left(\sum_{j=1}^{n} a_{ij} x_j \right). \tag{3.6}$$

The maximum and minimum activity levels for a partial solution are defined as:

$$MaxPAL_i = \max_{x_j \in \{0,1\}} \left(\sum_{j \in V^{\#}} a_{ij} x_j \right) + \sum_{j \notin V^{\#}} a_{ij} x_j \,, \tag{3.7}$$

$$MinPAL_i = \min_{x_j \in \{0,1\}} \left(\sum_{j \in V^{\#}} a_{ij} x_j \right) + \sum_{j \notin V^{\#}} a_{ij} x_j \,, \tag{3.8}$$

Let the normalized constraint coefficients be

$$a_{ij}^{normalized} = \frac{a_{ij}}{\overline{a}_i} \tag{3.9}$$

where

$$\overline{a}_i = \frac{\sum_{j=1}^{n} |a_{ij}|}{|\{j : a_{ij} \neq 0\}|} \tag{3.10}$$

3.1 Optimum satisfiability problem

The optimum satisfiability problem is an optimization problem assigning values to Boolean variables to satisfy a Boolean expression (Davoine, Hammer, and Vizvári 2003). This problem is an extension of a well-known Boolean satisfiability problem. Boolean satisfiability problem answers a question whether there exists an assignment of Boolean variables that satisfies a given formula. The optimum satisfiability assigns profit for setting each variable to 1 and answers the question what is the most profitable assignment that satisfies the given formula.

The problem itself was formulated as BIP by da Silva, Hvattum, & Glover, 2020. A Boolean formula have to be presented in a disjunctive normal form $f = T_1 \vee \ldots \vee T_m$ where T_i is a product of non-negated and negated variables.

$$\max z = \sum_{j=1}^{n} c_j x_j, \tag{3.11}$$

$$\sum_{j \in A_i} x_j - \sum_{j \in B_i} x_j \leq |A_i| - 1, \qquad i \in \{1, \ldots, m\}, \tag{3.12}$$

$$x_j \in \{0,1\}, \qquad j \in \{1, \ldots, n\} \tag{3.13}$$

where A_i and B_i are sets of non-negated and negated variables respectively in clause T_i, c_i is the profit from making the variable x_i true.

3.2 Multidemand multidimensional knapsack problem

The multidemand multidimensional knapsack problem (MDMKP) is a version of a well-known knapsack problem. Knapsack problem is used to determine a set of items maximizing profit while the total weigh of items does not exceed the limit. Multidimensional knapsack problem adds more “knapsack” constraints that are similar to the weight constraint. Multidemand knapsack problem introduces “covering” constraints which are opposite to “knapsack” constraints. Covering constraint require the sum of parameters in a dimension to be greater or equal to the limit.

The multidemand multidimensional knapsack problem (MDMKP) has the following formulation (Cappanera and Trubian 2005):

$$\max Z = \sum_{j=1}^{n} c_j x_j \,, \tag{3.14}$$

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \qquad i \in \{1, \ldots, m\}, \tag{3.15}$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i, \qquad i \in \{m+1, \ldots, m+q\}, \tag{3.16}$$

$$x_j \in \{0,1\}, \qquad j \in \{1, \ldots, n\} \tag{3.17}$$

c_j is the profit from including item j in the solution, a_{ij} is the size of the item j in the dimension i, b_i is the limit for the dimension i. Problem has m “knapsack” constraints and q “covering” constraints.

3.3 Multiple-choice multidimensional knapsack problem

Multiple-choice multidimensional knapsack problem is another version of a knapsack problem. The difference from the multidimensional knapsack problem is that in this version there are n disjoint sets of items $G_1, \ldots, G_n$ and exactly one item from each set have to be selected.

The mathematical formulation of this problem is the folowing:

$$\max Z = \sum_{i=1}^{n} \sum_{j \in G_i} c_{ij} x_{ij} \,, \tag{3.18}$$

$$\sum_{i=1}^{n} \sum_{j \in G_i} a_{ijk} x_{ij} \leq b_k \,, \qquad k \in \{1, \ldots, m\}, \tag{3.19}$$

$$\sum_{j \in G_i} x_{ij} = 1, \qquad i \in \{1, \dots, n\} \tag{3.20}$$

$$x_{ij} \in \{0,1\}, \qquad i \in \{1, \dots, n\}, j \in G_i. \tag{3.21}$$

This problem has equality constraints that are hard to satisfy and it can be hard to obtain a feasible solution for this class of problems.

3.4 Max-cut problem

The max-cut problem is defined on a weighted undirected graph $G = (V, E)$. The goal of the problem is to divide the nodes of the graph into two sets $\{S \subset V, V \setminus S\}$ so that the sum of weights for all egdes between the sets is maximized. The problem is not originally formulated as BIP but a BIP formulation is proposed by Lars Magnus Hvattum.

$$\max Z = \sum_{(u,v) \in E}^{n} w_{uv} y_{uv}, \tag{3.22}$$

$$y_{uv} - x_u - x_v \leq 0, \qquad (u, v) \in E: w_{uv} \geq 0 \tag{3.23}$$

$$y_{uv} + x_u + x_v \leq 2, \qquad (u, v) \in E: w_{uv} \geq 0 \tag{3.24}$$

$$-y_{uv} - x_u + x_v \leq 0, \qquad (u, v) \in E: w_{uv} < 0 \tag{3.25}$$

$$-y_{uv} + x_u - x_v \leq 0, \qquad (u, v) \in E: w_{uv} < 0 \tag{3.26}$$

$$x_u \in \{0, 1\} \qquad u \in V \tag{3.27}$$

$$y_{uv} \in \{0, 1\} \qquad (u, v) \in E \tag{3.28}$$

Where x_u shows whether the node u belongs to S, and $y_{uv} = 1$ if and only if nodes u and v belong to different sets.

The problem is not a typical BIP so it can be hard to obtain a good solution for this class of problems.

4.0 Solution methods

GRASP is usually based on a greedy construction heuristic followed by an improvement heuristic which can be a local search a more advanced technique such as variable neighborhood search. In this chapter the building blocks of GRASP are discussed with their pros and cons.

4.1 Greedy Construction Heuristic

The goal of a greedy construction heuristic is to create a solution for an instance of a problem by sequentially setting all the variables to a value. One idea can be to select the order of variables and their values by random. However, the goal of solving a BIP instance is to get a solution that is both feasible and has a high value of the objective function. This means that the greedy construction heuristic have to take into account the objective function coefficient corresponding to the variable (a value of a variable) and the constraints coefficients corresponding to the variable (a weight of a variable).

Let $V^{\#}$ be the set containing all the unassigned variables. Then the construction heuristic will start from an empty solution where $V^{\#}$ contains all the variables and end with an empty $V^{\#}$.

The pseudocode for a greedy construction heuristic is presented below:

1. $V^{\#} \coloneqq \{1, \dots, n\}$ // make all variables unassigned
2. $\boldsymbol{while}\ V^{\#} \neq \emptyset\ \boldsymbol{do}$ // while there are unassigned variables
3. $\quad Choose\ a\ variable\ j^*$ //chose a variable from $V^{\#}$
4. $\quad x_{j^*} \coloneqq 0\ or\ 1$ // set a value for the selected variable
5. $\quad V^{\#} \coloneqq V^{\#} \setminus \{j^*\}$ // mark variable as assigned
6. $\boldsymbol{end\ while}$

During each step the variable to assign a value is selected greedily following a criterion. This criterion should be based on a combination of the value of the variable and the weight of the variable.

During the first step of the algorithm all the variables becomes unassigned. But in order to be able to evaluate solution with partially assigned variables it is important to define what unassigned means. For the binary problems, it is common to assume that unassigned variable is set to zero (Vianna and Arroyo 2004). And this is the only implemented part by now, so hopefully I will write more on this topic later.

Line 3 of the pseudocode requires selecting the best possible variable. In order to do this there is a need to calculate rating of the variables that shows how good a possible assignment of each variable is and then select the variable with the highest rating. There are different ways to calculate rating described later. For deciding how good a possible assignment of the variable can be both influence on the constraints and influence on the objective function value should be taken into consideration. The influence on the objective function value is trivial but the influence on the constraints consists of the influences on every constraint. Weight is an artificial measure introduced to quantify the influence of the variable on the constraints.

4.1.1 Calculating weight

The weight of the variable is a measure of space occupied by the variable in the constraints if set to 1. There are different options to calculate this weight.

The easiest way to combine coefficients from different constraints is to take a sum of all coefficients. This can introduce a problem of different scales for different constraints e. g. :

$$x_1 + x_2 \leq 1,$$
$$10x_3 + 10x_4 \leq 10.$$

The sense is the same for both constraints, however the coefficients in the left-hand side differ. This can be solved by using normalized coefficients, which are introduced in chapter 3 equation (3.9).

As the problem has both lower bound constraints and upper bound constraints they should be treated differently. The weight of a variable is positive if setting a variable to 1 reduces free space (the difference between the bound and the activity level) and negative if setting a variable to 1 increases free space.

The formula for static weight is

$$weight_j^{static} = \sum_{\{i: a_{ij} \neq 0, i \in UB\}} a_{ij}^{normalized} - \sum_{\{i: a_{ij} \neq 0, i \in LB\}} a_{ij}^{normalized}, \qquad (4.1)$$

Where UB is a set of all upper bound constraints, LB is a set of all lower bound constraints.

The downside of this method is that the weight is not changing according to the change in the activity level of constraints and it’s not related to free space in the constraint. If a variable has the same coefficient for two constraints, but the activity level of one constraint is close to the bound and another constraint has a lot of free space the contribution to the weight of a variable would be the same for both constraints.

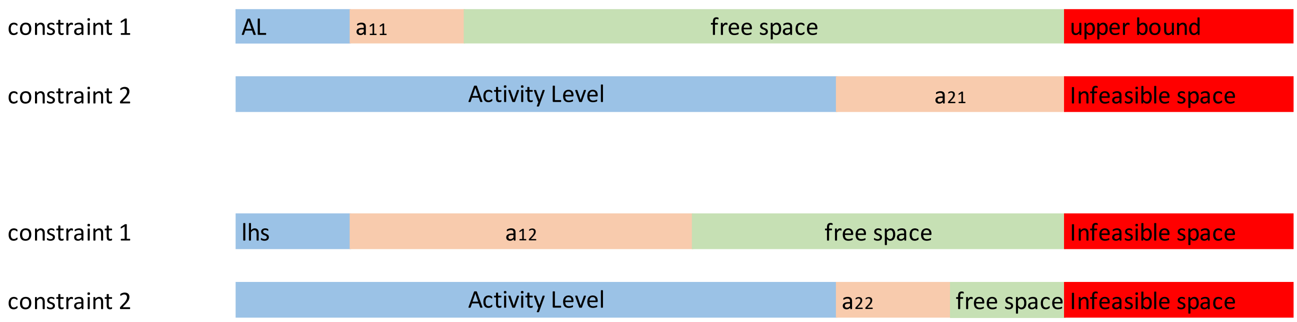

Figure 4.1 Knapsack problem example

In the example above (a knapsack problem with positive coefficients) the static weight of the first variable is lower, but setting this variable to one can lead to the second constraint being very close to bound and the impossibility of setting another variable to one. However, setting the second variable to one seems more promising because it leaves space in the second constraint to set more variables to one in order to increase the objective function value.

The advantage of this method is low computational complexity. The weight can be calculated once and does not change during the solution.

The problem of the static weight approach can be solved by using an importance of each constraint.

$$weight_j^{dynamic} = \sum_{\{i:a_{ij}\neq 0, i\in UB\}} a_{ij} * importance_i - \sum_{\{i:a_{ij}\neq 0, i\in LB\}} a_{ij} * importance_i \qquad (4.2)$$

This will allow to give different importance to different constraints depending on the difference between the bound and the activity level (free space) for every constraint.

$$space_i = \begin{cases} b_i^u - AL_i, & i \in UB \\ AL_i - b_i^l, & i \in LB \end{cases} \qquad (4.3)$$

The upper bound constraint is discussed below. The reasoning for the lower bound is similar. Lower bound constraint could be changed to the upper bound constraints by multiplying by -1. This results in the different signs for different part of formulas. Equality constraints are treated as two constraints with different signs.

Non-zero coefficients in all the constraints for a variable contribute to its weight. And this contribution depends on the coefficient itself and on free space in the constraint. The less free space the constraint has the higher importance it will have.

If a constraint has a lot of free space, it’s not quite important right now and the contribution of the coefficient won’t be huge.

For the positive coefficient if the constraint is currently infeasible making it even more infeasible by setting a variable to 1 is something which should be avoided. This will work also for a negative coefficient. It’s quite important to make a currently infeasible constraint (with high importance) more feasible. The contribution to the weight should be huge (because of importance) and negative because of the coefficient. Low weight is beneficial for the variable to be selected during a construction step. The plot the importance of the constraint will look similar to the following:

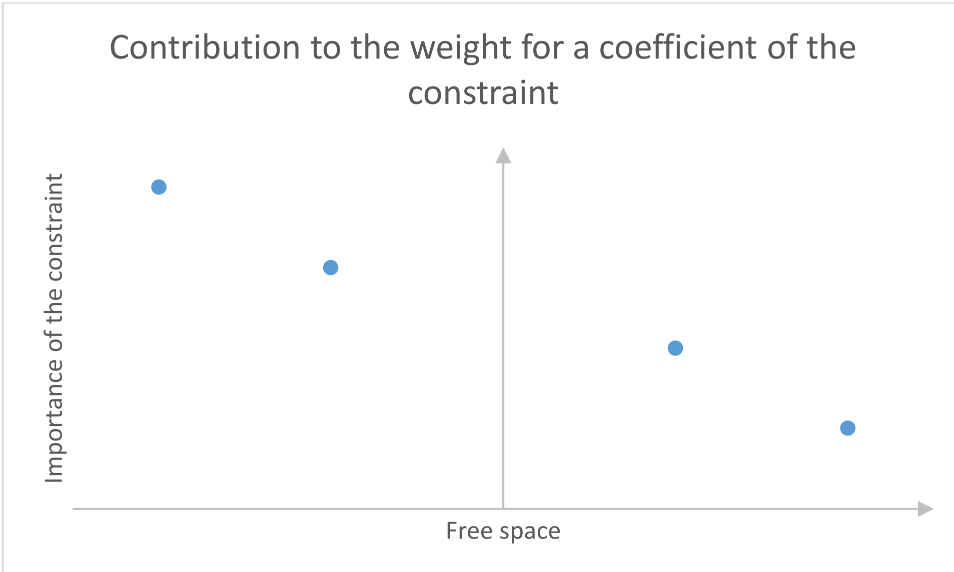

Figure 4.2 Plot of the importance dependency on free space

The initial idea is to use an inverse of free space.

This will work for the constraints, which are satisfied from the beginning. But using an inverse can be not the best choice here because it's plot is similar to the required only for the positive side.

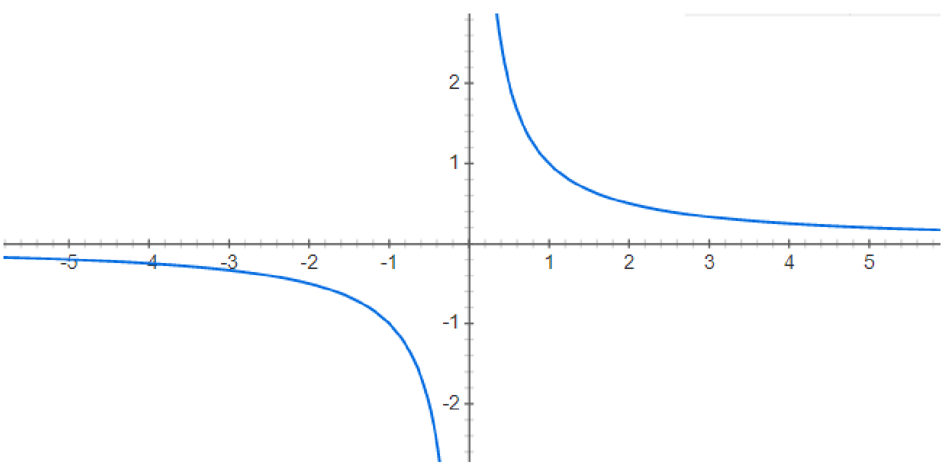

Figure 4.3 Plot of the function y = 1 / x

A more suitable choice here can be a modified sigmoid function:

$$importance_i = 1 - \frac{1}{1 + e^{-space_i}} \tag{4.4}$$

It has a following plot:

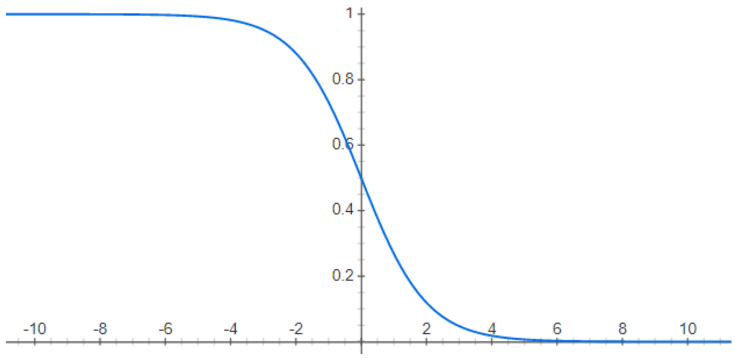

Figure 4.4 Importance plot

The problem here is that $space_i$ can be quite large (or negatively large) and for values greater than for example 10 the difference in importance is extremely small. In order to solve this the values of $space_i$ have to be normalized somehow.

$\frac{space_i}{MaxAL_i - MinAL_i}$ is a number from -1 to 1. Because $MaxAL_i - MinAL_i \geq |space_i|$ and $MaxAL_i - MinAL_i > 0$ if the constraint has at least one non-zero coefficient.

It can be called normalized space. It solves a problem of space being too large but introduces a problem of space being too small. This can lead to the insensible influence of the importance of different constraints. In order to make the influence of importance higher normalization can be used.

$$scale(x) = \frac{x - \mu}{\sigma}, \tag{4.5}$$

Where $x = (x_1, x_2, \dots, x_n)$, $\mu = \frac{\sum_{j=1}^{n} x_j}{n}$, $\sigma = \sqrt{\frac{\sum_{j=1}^{n}(x_j - \mu)^2}{n}}$.

$$importance^{scaled} = scale(importance) - \min\big(scale(importance)\big) + \delta \tag{4.6}$$

Subtracting the minimum element and adding delta makes $importance^{scaled}$ greater than zero because a constraint should have a positive importance. Delta can be different.

4.1.2 Calculating rating

In order to choose a variable to set a value greedily we need a rating of all variables. As stated earlier this rating should depend on the weight of the variable and its objective function value.

It’s beneficial to have a high objective function coefficient and low weight. This means that rating can be obtained by multiplication of the objective function coefficient by a measure, which is opposite to weight. An inverse of the weight won’t work because of the reasons described in the previous section. It’s possible to use $1 - sigmoid(weight)$ for the weight term of the product and $sigmoid(scale(c))$ for the other part. It’s important to use $sigmoid(scale(c))$ and not just $scale(c)$ to avoid multiplying $weight$ by 0 (in this case we lose all the information about the weight). Scaling is important to avoid extremely large (small) values that becomes close to 1 (0) after the use of sigmoid. This leaves us with

$$raiting_j^{sigmoid} = \left(\frac{1}{1+e^{-scale(c)_j}}\right) * \left(1 - \frac{1}{1+e^{-weight_j}}\right), \qquad (4.7)$$

The unassigned variable with the highest rating will be used at the next move of the construction heuristic.

Another approach to study is the sum instead of the multiplication of terms related to the objective function coefficient and weight. However objective function coefficients and weights have a different range and distribution. In order to make them comparable a normalization or scaling (statistical) can be used.

$$rating_j = scale(c)_j - scale(weight)_j \qquad (4.8)$$

Where $weight$ can be static or dynamic and c is a vector of objective function coefficients

Actually, multiplying or adding of the terms can lead to the loss of information (one term will be dominating all the time). This can be avoided by creating 2 different ratings for objective function contribution and weight. After two ratings are created each variable will have a rank in both of them. The sum of ranks in both ratings can be used to select the best variable at each step (The lower the sum is, the better the variable). To follow the fact that the best variable to assign has the highest rating a negative sum can be used.

$$rating_j = -rank(c_j) - rank(weight_j) \qquad (4.9)$$

Table 4.1 Sum of ranks example

rank	variable	weight	rank	variable	of coefficient	variable	sum of ranks
1	1	-4.25	1	2	10.5	1	1 + 4 = 5
2	3	0	2	3	7	2	4 + 1 = 5
3	4	2	3	4	3	3	2 + 2 = 4
4	2	14	4	1	-5	4	3 + 3 = 6

Variable 3 is best to assign in this example.

However, this approach is not perfect either. An issue here is the loss of the information. Now variables having almost identical weight would be one place away from each other. Variables that have a huge difference in weight can also be neighbors in the rating table if there are no other variables with the weight between the weights of the selected variables.

4.1.3 Accepting move

After a variable is selected during a step of a construction heuristic its value have to be determined. One of the following approaches can be used.

Here I understood that all the criteria that I have try to set a variable to 1 even if has a negative objective function coefficient. This is also something I should think of.

The method «Do not make worse» is based on the feasibility of constraints. The idea is to set a variable to 1 if this won't change any constraint from being feasible to being infeasible. If there is a constraint which is not violated but becomes violated after setting a variable to 1 then a variable value will become zero. This approach helps to stay in the feasible space but it might fail to find a good solution because sometimes it's beneficial to go into infeasible part of the solution space in order to find a better solution later.

The method «Be able to recover» is based on the possible feasibility of constraints.

Minimum and maximum activity levels for a partial solution show the minimum and maximum values that can be reached by the left-hand side of the constraint choosing different values only for unassigned variables. If $MinPAL_i$ will become higher than the upper bound value this means that the constraint cannot be satisfied. The same is true for a situation where $MaxPAL_i$ becomes lower than the lower bound value. These two situations have to be avoided. And this is the idea of this method.

A variable would be set to 1 if and only if after making this variable equal to 1 $MinPAL_i \leq b_i^u$ and $MaxPAL_i \geq b_i^l$.

Both approaches described above set a variable to 1 if it is possible. This can be improved by changing the criteria from possible to beneficial. Now beneficial should be defined. Setting a variable to 1 can be beneficial for the solution if the assignment is possible following a criteria described above and the assignment either improves the objective function value or it improves the state of the constraints. Improving the objective function value means having a positive objective function coefficient for the assigned variable. Weight of the variable is correlated with the change of the state of constraints. A negative value of weight can be a sign of the improvement in the state of the constraints. This approach can help to avoid setting a variable to 1 if this action is possible but it will reduce the objective function value and it will reduce free space in constraints.

4.1.4 Dealing with infeasibility

The goal of solving a problem is to get a feasible solution. A feasible solution even with a low objective function value is better than infeasible solution with the maximum possible objective function value.

It can be beneficial to encourage the construction heuristic to try not to end up in infeasible space. This can be achieved by setting a higher importance to the constraints which are currently infeasible. This will contribute to weight of the variables.

Then for the static weight the formula would become

$$weight_j^{static} = \sum_{\{i:a_{ij}\neq 0, i\in UBF\}} a_{ij}^{normalized} + \alpha * \sum_{\{i:a_{ij}\neq 0, i\in UBI\}} a_{ij}^{normalized} - \\ - \sum_{\{i:a_{ij}\neq 0, i\in LBF\}} a_{ij}^{normalized} - \alpha * \sum_{\{i:a_{ij}\neq 0, i\in LBI\}} a_{ij}^{normalized} \quad (4.10)$$

Where

- UBF stands for upper bound constraints which are currently satisfied (feasible)
- UBI – stands for upper bound constraints which are not currently satisfied (infeasible)
- LBF – stands for lower bound constraints which are currently satisfied (feasible)
- LBI – stands for lower bound constraints which are not currently satisfied (infeasible)

The same idea can be applied to the dynamic weight:

$$weight_j^{dynamic} = \sum_{\{i:a_{ij}\neq 0, i\in UBF\}} a_{ij} * importance_i + \alpha * \sum_{\{i:a_{ij}\neq 0, i\in UBI\}} a_{ij} * importance_i - \\ - \sum_{\{i:a_{ij}\neq 0, i\in LBF\}} a_{ij} * importance_i - \alpha * \sum_{\{i:a_{ij}\neq 0, i\in LBI\}} a_{ij} * importance_i , \quad (4.11)$$

However, this step can be redundant for the dynamic weight because importance itself contains information about free space in the constraint which corresponds to feasibility. A better approach for finding a feasible solution is to give more importance to weight term in rating formula. Using α inside of the $weight$ formula is not that efficient because $weight$ is normalized later. A coefficient $infeasibility$ can be applied to the whole $weight$ term as

$$rating_j = scale(c)_j - infeasibility * scale(weight)_j, \quad (4.12)$$

The sum of ranks for the rating now will look like

$$rating_j = -rank(c_j) - infeasibility * rank(weight_j), \quad (4.13)$$

4.2 Comparing solutions

Both improvement heuristic as a part of GRASP and GRASP itself deal with the comparison of solutions. A high objective function value does not matter a lot if the solution is infeasible. This means that solution x' is better than solution x if x' has lower infeasibility. And infeasibility can be expressed as

$$violation = violation^{sum} + \beta * violation^{count}, \qquad (4.14)$$

where

$$violation^{sum} = \sum_{i=1}^{m}\left(\max\left(\sum_{j=1}^{n} a_{ij}^{normalized} x_j - \frac{b_i^u}{\overline{a}_i}, 0\right) + \max\left(\frac{b_i^l}{\overline{a}_i} - \sum_{j=1}^{n} a_{ij}^{normalized} x_j, 0\right)\right), \qquad (4.15)$$

$violation^{count}$ is a number of unsatisfied constraints.

If solutions have the same infeasibility, then a solution with a higher objective function value is better.

4.3 Local search

A simple local search is implemented as the improvement part of GRASP algorithm. The pseudocode of the local search is presented below:

1: input: initial solution, x

2: input: neighborhood operator, N

3: **while** there is a $x' \in N(x)$ that is better than x **do**

4: Choose the best neighbor $x' \in N(x)$ that is better than x, and update $x := x'$.

5: **end while**

Currently local search is implemented only for a double-flip neighborhood.

$$N^2(x) = \left\{x' : \sum_{j=1}^{n} |x_j - x'_j| = 2\right\} \qquad (4.16)$$

4.4 GRASP

The pseudocode of the resulting GRASP algorithm is presented below:

1. input: initial value for $infeasibility$ and $\Delta infeasibility$
2. input: criteria for running the local search
3. input: α parameter
4. **while** stopping criterion not met **do**
5. $V^{\#} := \{1, \ldots, n\}$

6. **while** $V^{\#} \neq \emptyset$ **do**
7. update $rating$
8. Find a reduced candidate list, $L^{RC} \subseteq V^{\#}$ consisting of α% best ranked variables according to $rating$
9. select randomly $j \in L^{RC}$
10. set a value to $x_j \coloneqq d$, according to accepting move criterion
11. $V^{\#} \coloneqq V^{\#} \setminus \{j\}$
12. **end while**
13. run local search for the solution x if the criteria for running the local search is met
14. update the best solution x' a better solution was found
15. update $infeasibility$
16. **end while**
17. run local search for the solution x' if the criteria for running the local search is met
18. return x'

There are several parameters that can be changed in the presented algorithm.

α parameter is used to control the size of the restricted candidate list. A low value of alpha corresponds to a small size of a restricted candidate list and to a weak influence of randomness. With 0 as a value for α the algorithm becomes a greedy construction. A large value of α leads to a large candidate list and a solution highly influenced by the randomness.

$infeasibility$ parameter is used as described previously. It shows how important is weight comparing to the objective function coefficient for the rating of a variable. High values of the $infeasibilty$ parameter make weight more important. And it more likely to find a feasible solution with weight being more important. $infeasibilty$ parameter is updated depending on the previous solution found. If during the last iteration of GRASP an infeasible solution was found then weight should be more important and $infeasibility$ is increased additively. If the last solution found was feasible that it makes sense to focus on getting a solution with a higher objective function value and $infeasibility$ is decreased.

$$infeasibility^{i+1} = \begin{cases} infeasibility^{i} + \Delta infeasibility, if\ x^{i}\ is\ infeasible \\ infeasibility^{i} - \Delta infeasibility, if\ x^{i}\ is\ feasible \end{cases} \quad (4.17)$$

Where $infeasibility^{i}$ is the value of the infeasibility used on the i^{th} iteration of GRASP and x^{i} is the solution found during the i^{th} iteration of GRASP.

There are different options to run the local search.

- Use local search for every solution found by GRASP (always run local search on line 13 of the pseudocode)

- Use local search for a solution found by GRASP if it is better than the solution that was improved to get the current best solution (run local search on line 13 of the pseudocode if the criteria is met)
- Use local search only for the best solution found by GRASP (run local search on line 17 of the pseudocode)
- Do not use local search

5.0 Results

To evaluate whether GRASP can be used to find solutions for the BIP, the algorithm described in the previous chapter were implemented using c++ programming language. Computational tests were run using a laptop with 2.60GHz i7-4720HQ CPU with 8 GB RAM, running Windows 10.
Tables with the results of testing are presented further in this chapter. The result of a test is usually the objective function value, the number of violated constraints and the normalized sum of violations. The majority of solutions found are feasible. In order to avoid columns in a table containing only zeroes (for the number of violated constraints and the normalized sum of violations) the results are presented in a following way:

- Objective function value, if the solution is feasible
- Objective function value (number of violations; normalized violation sum)

5.1 Test instances

Both training set and test set consist of different instances of BIP from literature. The instances for the training set and for the test set were selected from literature and they represent different problem classes.
Training set is used to decide which approaches work better for different parts of the GRASP algorithm and of the greedy construction algorithm. The same instances are used for obtaining the best parameter values for GRASP. Training set consists of the following instances:

Table 5.1 Training set instances

Instance	Type	Number of variables	Number of constraints	Non-zero coefficients	Source
100-5-1-0-0	MDMKP	100	6	600	(Cappanera and Trubian 2005)
100-5-2-0-0	MDMKP	100	7	700	(Cappanera and Trubian 2005)
100-5-5-0-0	MDMKP	100	10	1000	(Cappanera and Trubian 2005)
100-5-5-1-0	MDMKP	100	10	1000	(Cappanera and Trubian 2005)

250-10-1-0-0	MDMKP	250	11	2750	(Cappanera and Trubian 2005)
250-10-5-0-0	MDMKP	250	15	3750	(Cappanera and Trubian 2005)
250-10-10-0-0	MDMKP	250	20	5000	(Cappanera and Trubian 2005)
g1	MAXCUT	19976	38352	115056	(Helmberg and Rendl 2000)
g14	MAXCUT	5494	9388	28164	(Helmberg and Rendl 2000)
sg3dl051000	MAXCUT	500	750	2250	(Festa et al. 2002)
sg3dl101000	MAXCUT	4000	6000	18000	(Festa et al. 2002)
100-5-01	MKP	100	5	500	(Chu and Beasley 1998)
250-10-01	MKP	250	10	2500	(Chu and Beasley 1998)
500-30-01	MKP	500	30	15000	(Chu and Beasley 1998)
I1	MMKP	25	10	133	(Khan et al. 2002)
INST01	MMKP	500	60	5022	(Khan et al. 2002)
rn50m30t4s0c0num0	OptSAT	50	30	120	(Davoine, Hammer, and Vizvári 2003)

Test set contains 4 problem classes with 15 instances each. The test set includes the instances used by Bentsen and Hvattum 2020 to perform a comparison of GRASP and the methotd created by the authors.

Table 5.2 Test set instances

Instance	Type	n	m	non-Zero	Source
I5	MMKP	250	35	2508	(Khan et al. 2002)
I9	MMKP	2000	210	20009	(Khan et al. 2002)
I11	MMKP	3000	310	30027	(Khan et al. 2002)
I13	MMKP	4000	410	40050	(Khan et al. 2002)
INST01	MMKP	500	60	5022	(Khan et al. 2002)
INST03	MMKP	600	70	6024	(Khan et al. 2002)
INST07	MMKP	800	90	8009	(Khan et al. 2002)
INST18	MMKP	5600	290	61600	(Khan et al. 2002)
INST20	MMKP	7000	360	77000	(Khan et al. 2002)
INST21	MMKP	1076	210	10763	(Shojaei et al. 2013)
INST24	MMKP	584	140	21675	(Shojaei et al. 2013)
INST28	MMKP	1643	310	16439	(Shojaei et al. 2013)
RTI09	MMKP	158	40	1568	(Shojaei et al. 2013)
RTI12	MMKP	241	50	2448	(Shojaei et al. 2013)
RTI13	MMKP	295	60	2954	(Shojaei et al. 2013)
100-5-5-1-0	MDMKP	100	10	1000	(Cappanera and Trubian 2005)
100-10-5-1-0	MDMKP	100	15	1500	(Cappanera and Trubian 2005)
100-30-15-1-10	MDMKP	100	45	4500	(Cappanera and Trubian 2005)
100-30-30-0-1	MDMKP	100	60	6000	(Cappanera and Trubian 2005)
100-50-10-1	MDMKP	100	51	5100	(Cappanera and Trubian 2005)
100-50-q-1	MDMKP	100	51	5100	(Cappanera and Trubian 2005)
100-100-25-1	MDMKP	100	101	10100	(Cappanera and Trubian 2005)
100-100-q-1	MDMKP	100	101	10100	(Cappanera and Trubian 2005)
250-5-2-0-0	MDMKP	250	7	1750	(Cappanera and Trubian 2005)
250-10-1-0-14	MDMKP	250	11	2750	(Cappanera and Trubian 2005)
250-30-30-0-0	MDMKP	250	60	15000	(Cappanera and Trubian 2005)
500-5-5-0-14	MDMKP	500	10	5000	(Cappanera and Trubian 2005)
500-10-10-1-0	MDMKP	500	20	10000	(Cappanera and Trubian 2005)

500-30-15-1-0	MDMKP	500	45	22500	(Cappanera and Trubian 2005)
500-30-30-0-0	MDMKP	500	60	30000	(Cappanera and Trubian 2005)
lmhn1000m5000num1	OptSat	1000	5000	15000	(da Silva, Hvattum, and Glover 2020)
lmhn1500m7500num1	OptSat	1500	7500	22500	(da Silva, Hvattum, and Glover 2020)
qn500m2500t2s0c0num0	OptSat	500	2500	5000	(Davoine, Hammer, and Vizvári 2003)
qn500m5000t2s0c0num0	OptSat	500	5000	10000	(Davoine, Hammer, and Vizvári 2003)
qn1000m10000t2s0c0num0	OptSat	1000	10000	20000	(Davoine, Hammer, and Vizvári 2003)
rn200m1000t10s0c0num0	OptSat	200	1000	10000	(Davoine, Hammer, and Vizvári 2003)
rn200m1000t10s0c25num4	OptSat	200	1000	10000	(Davoine, Hammer, and Vizvári 2003)
rn200m1000t40s20c0num0	OptSat	200	1000	40187	(Davoine, Hammer, and Vizvári 2003)
rn500m1000t25s0c0num4	OptSat	500	1000	25000	(Davoine, Hammer, and Vizvári 2003)
rn500m1000t25s0c25num4	OptSat	500	1000	25000	(Davoine, Hammer, and Vizvári 2003)
rn500m1000t25s0c50num0	OptSat	500	1000	25000	(Davoine, Hammer, and Vizvári 2003)
rn500m1000t100s50c0num0	OptSat	500	1000	99511	(Davoine, Hammer, and Vizvári 2003)
rn500m1000t100s50c25num0	OptSat	500	1000	100798	(Davoine, Hammer, and Vizvári 2003)
rn500m2500t25s0c25num4	OptSat	500	2500	62500	(Davoine, Hammer, and Vizvári 2003)
rn500m2500t25s0c50num0	OptSat	500	2500	62500	(Davoine, Hammer, and Vizvári 2003)

g5	MaxCut	19976	38352	115056	(Helmberg and Rendl 2000)
g15	MaxCut	5461	9322	27966	(Helmberg and Rendl 2000)
g25	MaxCut	21990	39980	119940	(Helmberg and Rendl 2000)
g35	MaxCut	13778	23556	70668	(Helmberg and Rendl 2000)
g45	MaxCut	10990	19980	59940	(Helmberg and Rendl 2000)
g50	MaxCut	9000	12000	36000	(Helmberg and Rendl 2000)
g54	MaxCut	6916	11832	35496	(Helmberg and Rendl 2000)
sg3dl053000	MaxCut	500	750	2250	(Festa et al. 2002)
sg3dl105000	MaxCut	4000	6000	18000	(Festa et al. 2002)
sg3dl144000	MaxCut	10976	16464	49392	(Festa et al. 2002)
sg3dl1410000	MaxCut	10976	16464	49392	(Festa et al. 2002)
toursg3-8	MaxCut	2048	3072	9216	7th DIMACS Implementation Challenge
toursg3-15	MaxCut	13500	20250	60750	7th DIMACS Implementation Challenge
tourspm3-8-50	MaxCut	2048	3072	9216	7th DIMACS Implementation Challenge
tourspm3-15-50	MaxCut	13500	20250	60750	7th DIMACS Implementation Challenge

5.2 Approaches testing

In this section the results for testing different approaches for different parts of the greedy construction algorithm and of the GRASP are presented. To test different approaches for one part of the algorithm (e.g., weight calculation or rating calculation) the approaches for all the other parts are fixed and then the results for different approaches are compared to select the best one.

The approaches for calculating weight, rating and selecting a value are tested as a part of greedy construction algorithm. Using GRASP can lead to results influenced by the randomization and will not be the evidence of advantages of an approach.

For some instances, obtaining a feasible solution is harder than for other instances. That is why during testing different approaches for calculating weight, rating and selecting a value each instance was solved using the greedy construction heuristic using different values for the $infeasibility$ parameter.

Then the best solution was selected as the result of the test. The values for $infeasibility$ are $\{ 0.5, 1, 2, 3, 5 \}$.

Several values are used for the $infeasibility$ parameter for several reasons. $infeasibility$ parameter has a different influence on the result in different approaches, so it is impossible to select one value for all the approaches. Selecting 1 as a value will result in getting mostly infeasible solutions. Later the greedy construction will be used as a part of GRASP to obtain feasible solutions (or at least solutions that are close to feasible) with the help of the $infeasibility$ parameter. So, using 1 as a value for the $infeasibility$ for testing will not show the behavior of the algorithm that will be used later. That is why different values are used.

5.2.1 Weight

There are different approaches for calculating the weight.

- The weight can be calculated simply as a static weight
- Dynamic weight without the normalization of importance
- Dynamic weight with normalized importance

The fixed parts of the greedy construction algorithm are the following:

- A sum of normalized parts is used for rating calculation (equations (4.8) and (4.12))
- “Be able to recover” rule is used for selecting the value for a variable.
- The values for $infeasibility$ are $\{ 0.5, 1, 2, 3, 5 \}$.

The results for 3 different approaches follow:

Table 5.3 Results for the different approaches for calculating the weight

Instance	Static weight	Dynamic weight without normalization	Dynamic weight with normalization
100-5-1-0-0	26375	26375	28416
100-5-2-0-0	20804	20795	26534
100-5-5-0-0	15275	16119	16054
100-5-5-1-0	2215	2270	6954
250-10-1-0-0	54868	54868	55728
250-10-5-0-0	40064	40658	51269

250-10-10-0-0	38081	39373	46076
g1	0	0	0
g14	0	0	0
sg3dl051000	-173	-134	-187
sg3dl101000	-1494	-1229	-1500
100-5-01	23244	23306	24034
250-10-01	56325	56325	58474
500-30-01	109259	109259	113485
I1	147	147	147
INST01	8074	7853	8006
rn50m30t4s0c0num0	2912	2863	2863

All the methods are able to find a feasible solution for all the problems. The solutions for problems g1 and g14 are trivial but local search is not used, so it is hard to find better solutions. The fact that all the solutions are feasible allows to compare only the objective function values.

Table 5.4 Comparison of static and dynamic weight

Instance	Static weight	Dynamic weight without normalization	Difference
100-5-1-0-0	26375	26375	0
100-5-2-0-0	20804	20795	-9
100-5-5-0-0	15275	16119	844
100-5-5-1-0	2215	2270	55
250-10-1-0-0	54868	54868	0
250-10-5-0-0	40064	40658	594
250-10-10-0-0	38081	39373	1292
g1	0	0	0
g14	0	0	0
sg3dl051000	-173	-134	39
sg3dl101000	-1494	-1229	265

100-5-01	23244	23306	62
250-10-01	56325	56325	0
500-30-01	109259	109259	0
I1	147	147	0
INST01	8074	7853	-221
rn50m30t4s0c0num0	2912	2863	-49

Using dynamic weight is beneficial for all the problem classes except for MMKP and OptSAT. In general, it can be concluded that using $importance$ of each constraint is beneficial for the algorithm.

Table 5.5 Influence of the importance normalization

Instance	Dynamic weight without normalization	Dynamic weight with normalization	Difference
100-5-1-0-0	26375	28416	2041
100-5-2-0-0	20795	26534	5739
100-5-5-0-0	16119	16054	-65
100-5-5-1-0	2270	6954	4684
250-10-1-0-0	54868	55728	860
250-10-5-0-0	40658	51269	10611
250-10-10-0-0	39373	46076	6703
g1	0	0	0
g14	0	0	0
sg3dl051000	-134	-187	-53
sg3dl101000	-1229	-1500	-271
100-5-01	23306	24034	728
250-10-01	56325	58474	2149
500-30-01	109259	113485	4226
I1	147	147	0
INST01	7853	8006	153
rn50m30t4s0c0num0	2863	2863	0

Normalization improves the results for all problem classes except for MAXCUT.

It can be stated that the greedy algorithm works better in general when using dynamic weight with normalized importance.

5.2.2 Rating

As described in the previous chapter rating relies on two terms: weight and objective function coefficient. There are different approaches for calculating the rating.

- The sum of normalized terms with the use of $infeasibility$ coefficient to make one of them more important (equations (4.8) and (4.12))
- Sum of ranks of the terms with the use of $infeasibility$ coefficient to make one of them more important (equations (4.9) and (4.13))

The fixed parts of the greedy construction algorithm are the following:

- Dynamic weight with normalized importance is used to calculate weight
- “Be able to recover” rule is used for selecting the value for a variable.
- The values for $infeasibility$ are $\{ 0.5, 1, 2, 3, 5 \}$.

The results for the approach with the sum of normalized terms are presented in the previous section (Table 5.4). The result for the approach with the sum of ranks is below. If the solution is infeasible violation sum and violation count are presented in parentheses.

Table 5.6 Results for the sum of ranks

Instance	Objective function value
100-5-1-0-0	27590
100-5-2-0-0	23481
100-5-5-0-0	18722
100-5-5-1-0	3482
250-10-1-0-0	61534
250-10-5-0-0	46438
250-10-10-0-0	46142
g1	10268 (722; 722)
g14	1128
sg3dl051000	-181
sg3dl101000	-1496

100-5-01	23584
250-10-01	57508
500-30-01	112271
I1	143
INST01	8237 (1; 1)
rn50m30t4s0c0num0	2870

This approach fails to find a feasible solution for 2 instances from the training set while using the sum of normalized terms allows to find a feasible solution for all the instances from this set. The comparison of the objective function values for both methods is presented below (only for the instances where a feasible solution was found by both methods).

Table 5.7 Comparison of rating calculation approaches

Instance	Sum of ranks	Normalized sum	Difference
100-5-1-0-0	27590	28416	826
100-5-2-0-0	23481	26534	3053
100-5-5-0-0	18722	16054	-2668
100-5-5-1-0	3482	6954	3472
250-10-1-0-0	61534	55728	-5806
250-10-5-0-0	46438	51269	4831
250-10-10-0-0	46142	46076	-66
g14	1128	0	-1128
sg3dl051000	-181	-187	-6
sg3dl101000	-1496	-1500	-4
100-5-01	23584	24034	450
250-10-01	57508	58474	966
500-30-01	112271	113485	1214
I1	143	147	4
rn50m30t4s0c0num0	2870	2863	-7

Each approach works better than the opponent method in approximately half of the instances from the table 5.8. This fact does not provide an obviously better approach. But taking into account the fact that the sum of normalized terms approach is able to find a feasible solution for all the instances this approach is selected to be a part of the final algorithm. It can be beneficial in future to find a way to combine these two methods to get better results for some of the instances. And this is one of the questions for the further research.

5.2.3 Selecting a value

As described in the previous chapter there are two different approaches for selecting a value for a variable:

- “Do not make worse”
- “Be able to recover”
- “Beneficial instead of possible”

The fixed parts of the greedy construction algorithm are the following:

- Dynamic weight with normalized importance is used to calculate weight
- A sum of normalized parts is used for rating calculation (equations (4.8) and (4.12))
- The values for $infeasibility$ are $\{ 0.5, 1, 2, 3, 5 \}$.

The results for the “Be able to recover” approach are presented earlier (Table 5.4). The results for the “Do not make worse” approach follow:

Table 5.8 Results for the "Do not make worse" approach

Instance	Objective function value
100-5-1-0-0	28416
100-5-2-0-0	26534
100-5-5-0-0	16054
100-5-5-1-0	6954
250-10-1-0-0	55728
250-10-5-0-0	51269
250-10-10-0-0	46076
g1	0
g14	2120
sg3dl051000	-130
sg3dl101000	-956
100-5-01	24034

250-10-01	58474
500-30-01	113485
I1	147
INST01	8006
rn50m30t4s0c0num0	2863

The algorithm is able to find a feasible solution for every instance from the training set.

Table 5.9 Comparison of the approaches for selecting a value

Instance	Don't make worse	Be able to recover	Difference
100-5-1-0-0	28416	28416	0
100-5-2-0-0	26534	26534	0
100-5-5-0-0	16054	16054	0
100-5-5-1-0	6954	6954	0
250-10-1-0-0	55728	55728	0
250-10-5-0-0	51269	51269	0
250-10-10-0-0	46076	46076	0
g1	0	0	0
g14	2120	0	-2120
sg3dl051000	-130	-187	-57
sg3dl101000	-956	-1500	-544
100-5-01	24034	24034	0
250-10-01	58474	58474	0
500-30-01	113485	113485	0
I1	147	147	0
INST01	8006	8006	0
rn50m30t4s0c0num0	2863	2863	0

From the comparison it is clear that the approaches work quite similar. But for several instances “Do not make worse” approach shows better results. It means that there is no evidence that the algorithm benefits from the ability to make a constraint infeasible during the construction of a solution.

Setting a variable to 1 if it is possible and beneficial shows exactly the same result on the training set as setting a variable to 1 if it is possible. But for avoiding unnecessary setting variables with a negative impact on the solution “beneficial” approach will be used.

The approach “Do not make worse” with setting a variable to 1 if beneficial is selected as a part of the final algorithm.

5.2.4 Infeasibility

During testing the approaches described above several values were used as values for the $infeasibility$. The final greedy construction heuristic is tested with different values for the $infeasibility$ and the results are presented below.

Table 5.10 Results for the different values of infeasibility

infeasibility	0.5	1	2	3	5
100-5-1-0-0	30274(1;15.3133)	28416	23933	23820	23820
100-5-2-0-0	27655(2;16.4558)	26534	20240	20437	20437
100-5-5-0-0	18368(5;33.2733)	18289(1;0.192336)	16054	14608	14303
100-5-5-1-0	10502(5;33.3803)	6954	2746	2746	1671
250-10-1-0-0	69314(1;36.5684)	67243(1;6.35034)	55728	53734	48738
250-10-5-0-0	48757(5;78.3428)	51269	41165	38115	37050
250-10-10-0-0	47071(10;69.5535)	46076	39333	38057	37139
g1	0	0	0	0	0
g14	2120	0	0	0	0
sg3dl051000	-171	-171	-159	-130	-130
sg3dl101000	-1382	-1382	-1305	-973	-956
100-5-01	22522	24034	23065	23046	22647
250-10-01	57151	58474	54907	53532	52841
500-30-01	113485	111697	107024	103250	100117

I1	147	142	64	64	64
INST01	8583(20;20)	7699(16;16)	8006	7838	7708
rn50m30t4s0c0num0	2849	2814	2863	2863	2863

It is obvious that best solutions for different instances are obtained with different $infeasibility$ values. Usually, low values of the $infeasibility$ parameter lead to infeasible solutions. But for the majority of instances feasible solutions obtained with higher $infeasibility$ values are worse than feasible solutions obtained with lower $infeasibility$ values.

5.3 Parameters tuning

The GRASP presented in the previous chapter has several parameters. The parameters are α, $infeasibility$, $\Delta infeasibility$ and the criteria for running local search.

For tuning one of the parameters all the other parameters are fixed. The training set used for testing the greedy construction is used for tuning the parameters of GRASP.

The time limit for the algorithm is 5 minutes. Time limit for a local search is 30 seconds.

The results of testing the algorithm with different parameters and the best parameter values that are selected for the final algorithm are presented below.

5.3.1 α tuning

α is used to control the influence of randomness on a solution. The range of values is $[0, 1]$ with the algorithm being purely greedy if $\alpha = 0$ and a random construction if $\alpha = 1$. The values for testing include the extreme values and the values in between. The list of values for testing is $\{0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.1, 0.2, 0.5, 1\}$.

The fixed parts of GRASP are the following:

- $infeasibility = 2$
- $\Delta infeasibility = 0.1$
- Local search is used for every solution constructed

The results are presented in the tables below.

Table 5.11 Alpha testing results. Part 1

Instance \ α	0	0.01	0.02	0.03	0.04	0.05	0.06
100-5-1-0-0	30348	30348	30204	30299	30204	30120	30399
100-5-2-0-0	27884	27884	27686	27861	27903	27737	27590
100-5-5-0-0	21271	21271	21031	21304	21019	21032	20785
100-5-5-1-0	9507	9507	9701	9334	9390	9760	9667
250-10-1-0-0	65766	65820	65782	65576	65346	65862	65323
250-10-5-0-0	54265	54166	54516	54278	54901	54482	54250
250-10-10-0-0	50481	50921	50218	50411	50919	50990	50884

g1	0	2977	5346	7045	8519	9268	9746
g14	0	141	284	458	953	1250	1681
sg3dl051000	-141	-132	-123	-117	-115	-122	-107
sg3dl101000	-1500	-1467	-1454	-1437	-1437	-1430	-1420
100-5-01	24270	24270	24202	24285	24381	24231	24177
250-10-01	58807	58794	58871	58877	58725	58833	58729
500-30-01	114900	114773	114664	115111	114551	115192	114873
I1	167	167	167	167	167	173	173
INST01	10265	10306	10303	10323	10329	10307	10308
rn50m30t4s0c0num0	2912	2912	2912	2912	2912	2912	2912

Table 5.12 Alpha testing results. Part 2

Instance \ α	0.07	0.1	0.2	0.5	1
100-5-1-0-0	30271	30229	29828	28454	28350
100-5-2-0-0	27472	27787	27245	26327	26432
100-5-5-0-0	21064	20966	21186	21352	21137
100-5-5-1-0	9423	9634	9691	9737	9946
250-10-1-0-0	65525	64876	64370	61837	60643
250-10-5-0-0	54205	54618	53657	52932	51831
250-10-10-0-0	50293	50807	50234	49160	49298
g1	9718 (3;3)	9352 (9;9)	7842 (595;595)	10424 (4462;4462)	14465 (8731;8731)
g14	1549	2094	2505	2959 (562;562)	3705 (1461;1461)
sg3dl051000	-115	-105	-64	-32	-19
sg3dl101000	-1417	-1394	-1324	-855 (7;7)	-322 (179;179)
100-5-01	24182	24180	23779	23630	23966
250-10-01	58779	58457	58117	57743	57637

500-30-01	114867	114718	114496	113641	111496
I1	173	173	173	173	173
INST01	10336	10266	10254	10201	9975
rn50m30t4s0c0num0	2912	2921	2921	2960	2960

High values of α result in infeasible solution for some instances. No value gives the best result for all the instances. Different α values work better for different instances. In order to compare different results statistical standardizing is applied. The results for each instance are replaced with the standard score (number of standard deviations from mean). This allows to compare the results originally having different magnitudes.

Any infeasible solution is worse than a feasible that is why infeasible solutions should be treated separately and not by the objective function value. The standard scores for the infeasible solutions are set to -1. This mean that an infeasible solution is below average. The value can be lower but it does not change the result of the analysis.

Then the means of standard scores for each α value are calculated. These values show how different the result obtained with a specific α value is from the average result obtained in all tests. The plot of the values is presented below.

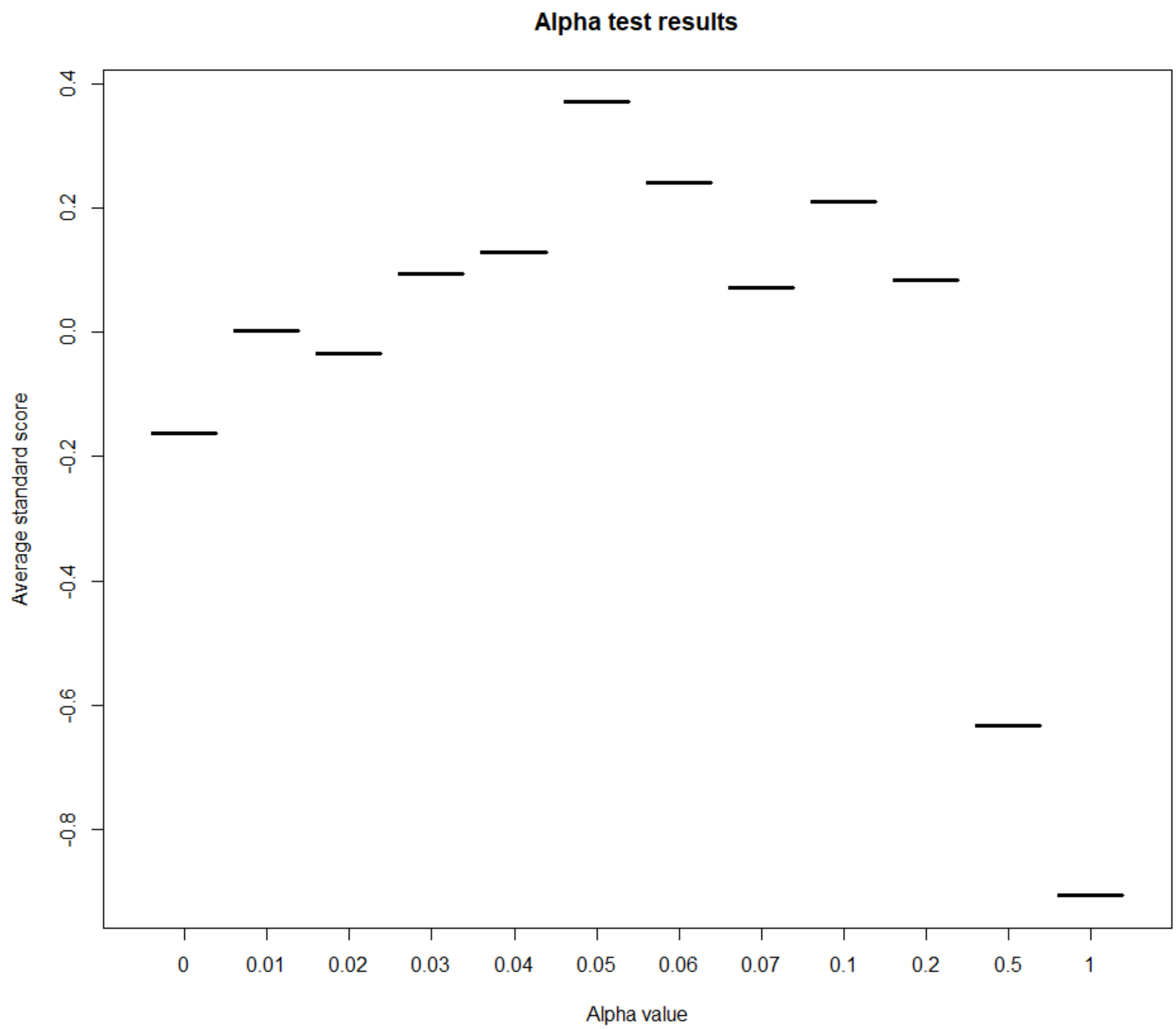

Figure 5.1 Plot of average standard scores for alpha testing

It is possible to see that all values in the range $[0.01, 0.2]$ show better results than the values 0 and 1 . This means that GRASP performs better than a greedy construction and a random construction.

The value 0.05 gives better results on average. The values close to 0.05 also give a good result. Extreme values or alpha perform purely. 0.05 is selected as the α value for the final algorithm.

5.3.2 $infeasibility$ tuning

The initial value for the $infeasibility$ has a high importance especially for the instances with a huge number of variables and constraints because the algorithm spends a lot of time during one iteration. That is why the number of iterations is low and the $infeasibility$ parameter does not change a lot during the runtime.

The possible values are from 0 to ∞. However, values higher than 10 do not make a huge difference. The testing was performed for the following list of values: $\{0, 0.5, 1, 2, 2.5, 3, 5, 7, 10\}$.

The fixed parts of GRASP are the following:

- $\alpha = 0.05$
- $\Delta infeasibility = 0.1$
- Local search is used for every solution constructed

The results are presented in the tables below.

Table 5.13 Results for infeasibility testing

Instance\ $infeasibility$	0	0.5	1	2	2.5	3	5	7	10
100-5-1-0-0	26832	27379	29707	30120	30121	30294	30294	30603	30429
100-5-2-0-0	26092	26092	26826	27737	27878	27649	27882	27751	28067
100-5-5-0-0	20360	20360	20360	21032	21214	21298	21190	21184	21117
100-5-5-1-0	9308	9308	9441	9760	9685	9760	9668	9559	9562
250-10-1-0-0	57388	59399	65281	65862	65698	65599	65197	65567	65565
250-10-5-0-0	48277	50475	52689	54482	54183	54088	54505	54956	54347
250-10-10-0-0	47784	47866	47880	50990	50976	50722	51057	50864	51123
g1	19176 (16050; 16050)	19174 (16042; 16042)	9627	9268	9442	9331	9337	9124	9208
g14	4483 (2617; 2617)	1762	2508	1250	832	861	840	855	838
sg3dl051000	-2	5	-5	-122	-115	-119	-121	-119	-117
sg3dl101000	-1042	-1282	-1070	-1430	-1393	-1378	-1385	-1389	-1394
100-5-01	21526	21618	23894	24231	24326	24281	24147	24191	24201
250-10-01	53013	57686	58641	58833	58722	58718	58569	58881	58738
500-30-01	107022	113548	114771	115192	114807	114756	115049	114990	114867
I1	173	173	173	173	173	173	173	173	173
INST01	10307	10282	10307	10307	10282	10307	10338	10310	10352

rn50m30t4s0c0num0	2794	2849	2871	2912	2912	2912	2912	2912	2912

The idea of using the average standard score is used to determine the best value.

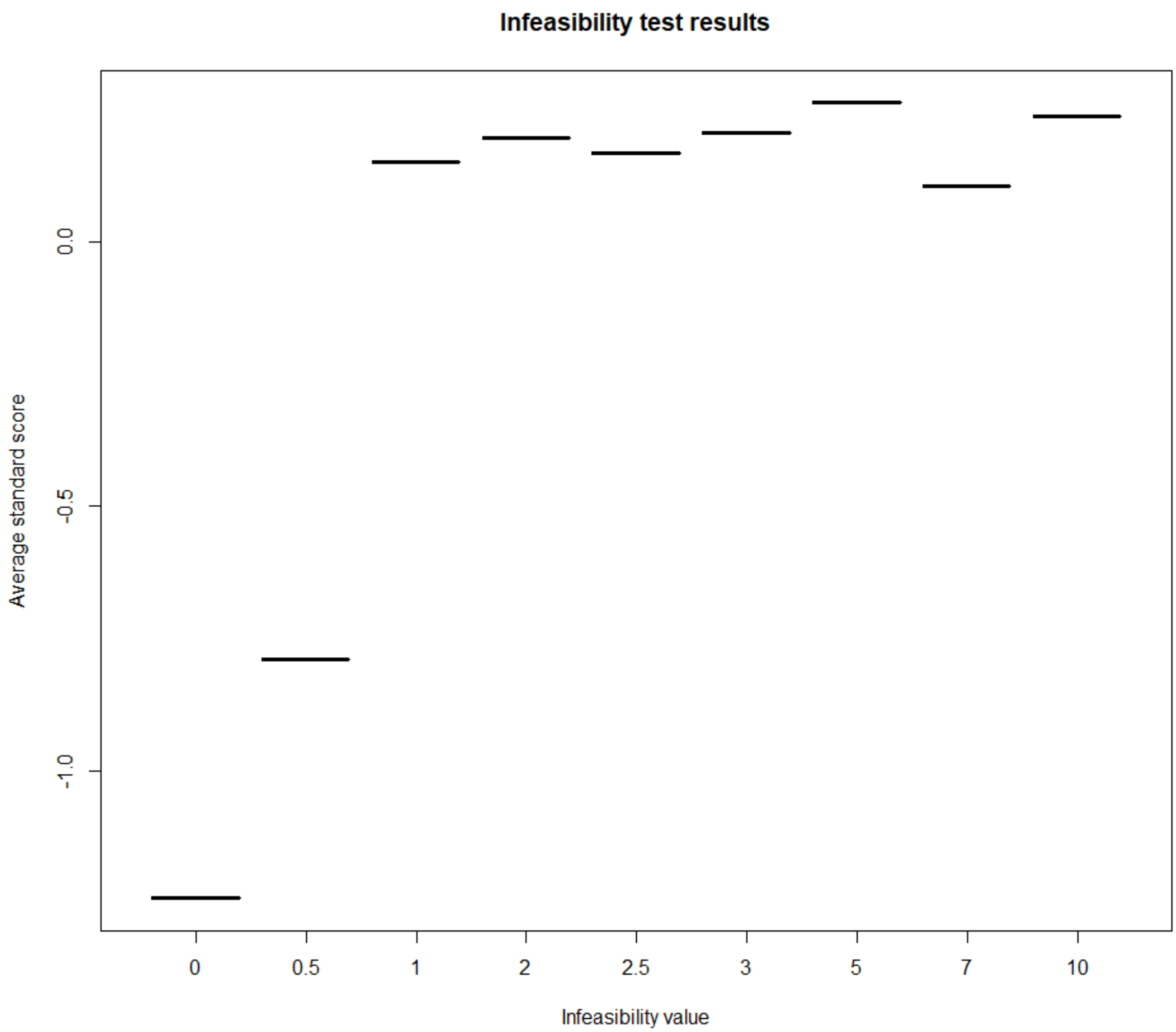

Figure 5.2 Plot of average standard scores for infeasibility testing

All the values starting from 1 perform similarly. But given that low values can lead to infeasible solutions for instances that are hard to solve a higher value should be selected. Value 5 gives the result on the training set that is slightly better than all the others and it seems to be high enough to obtain feasible solutions for the majority of instances. 5 is selected as the initial value for the $infeasibility$ parameter in the final algorithm.

5.3.3 $\Delta infeasibility$ tuning

$\Delta infeasibility$ controls how the $infeasibility$ changes. $\Delta infeasibility = 0$ leads to the same $infeasibility$ value for all the iterations. Higher values correspond to higher variability of $infeasibility$. The range of values for testing is $[0, 2]$. The values for testing are $\{0, 0.05, 0.1, 0.2, 0.3, 0.5, 1, 2\}$.

The fixed parts of GRASP are the following:

- $\alpha = 0.05$
- $infeasibility = 5$
- Local search is used for every solution constructed

The results are presented in the tables below.

Table 5.14 Results for delta infeasibility testing

	0	0.05	0.1	0.2	0.3	0.5	1	2
100-5-1-0-0	30665	30626	30294	30547	30235	30134	29837	29834
100-5-2-0-0	28185	27916	27882	27754	27360	27581	27360	27360
100-5-5-0-0	21547	21418	21190	21231	21162	20919	21212	20812
100-5-5-1-0	9555	9951	9668	9760	9616	9336	9391	9308
250-10-1-0-0	64118	65807	65197	65023	65341	65337	64542	64813
250-10-5-0-0	54870	54622	54505	54162	53915	53878	53626	52759
250-10-10-0-0	51173	50867	51057	51056	50736	50371	50544	49647
g1	9343	9255	9337	9433	9109	9386	9303	9087
g14	816	843	840	814	897	829	2313	2449
sg3dl051000	-119	-115	-119	-116	-116	-29	-6	-6
sg3dl101000	-1391	-1388	-1385	-1386	-1382	-1269	-1260	-1325
100-5-01	24007	24192	24147	24125	24121	24165	24199	23848
250-10-01	55403	58679	58569	58516	58675	58680	59016	58511
500-30-01	107456	114763	115049	114515	114895	114739	114218	113612
I1	159	173	173	173	173	173	173	173

INST01	10305	10295	10338	10312	10324	10326	10294	10271
rn50m30t4s0c0num 0	2912	2912	2912	2912	2912	2912	2912	2912

The approach with calculating average standard scores is used once again to determine the best value.

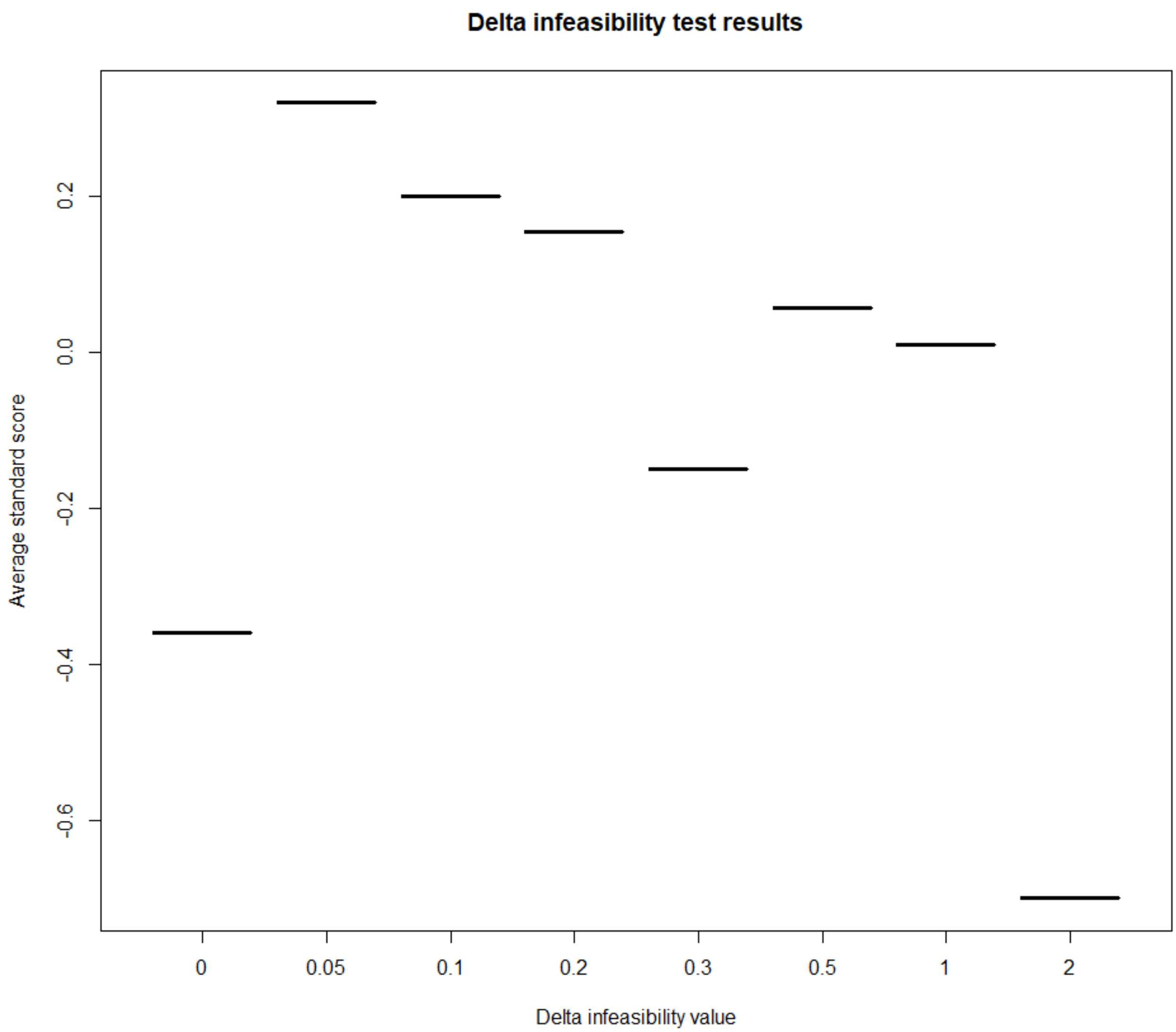

Figure 5.3 Plot of average standard scores for delta infeasibility testing

Value 0 gives bad results. This means that the approach of changing $infeasibility$ between GRASP iterations is profitable. The best value is 0.05. It is quite low and can lead to slow change of $infeasibility$. But as the algorithm starts from a high value of $infeasibility$ this should not cause the impossibility of the algorithm to obtain a feasible solution.

0.05 is selected as the value of $\Delta infeasibility$ for the final algorithm.

5.3.4 Local search tuning

Four different approaches of using local search are tested:

1. Running the local search only for the best solution found by GRASP
2. Running the local search for every solution found by GRASP
3. Running the local search for a solution found during the iteration of GRASP only if this solution is better than the solution that was improved to get the current best solution.
4. Not using local search

These approaches are tested because local search improves the solutions but takes some time to do so. Different approaches have different ratio of the time spend on constructing solution to the time spend on improving solutions.

The fixed parts of GRASP are the following:

- $\alpha = 0.05$
- $infeasibility = 5$
- $\Delta infeasibility = 0.05$

The results are presented in the table below.

Table 5.15 Results for local search testing

	1	2	3	4
100-5-1-0-0	30385	30626	30728	30385
100-5-2-0-0	27614	27916	27916	27614
100-5-5-0-0	20109	21418	21229	20059
100-5-5-1-0	9480	9951	9975	9480
250-10-1-0-0	66082	65807	66236	66098
250-10-5-0-0	54003	54622	54580	53136
250-10-10-0-0	49309	50867	50867	47805
g1	9255	9255	9255	9255
g14	865	843	856	865
sg3dl051000	-3	-116	-108	-45
sg3dl101000	-1369	-1388	-1369	-1369
100-5-01	24164	24192	24164	24164

250-10-01	58667	58679	58667	58080
500-30-01	114553	114763	114763	112709
I1	158	173	173	158
INST01	10169	10295	10301	9035
rn50m30t4s0c0num0	2912	2912	2912	2886

The plot of average standard scores is below.

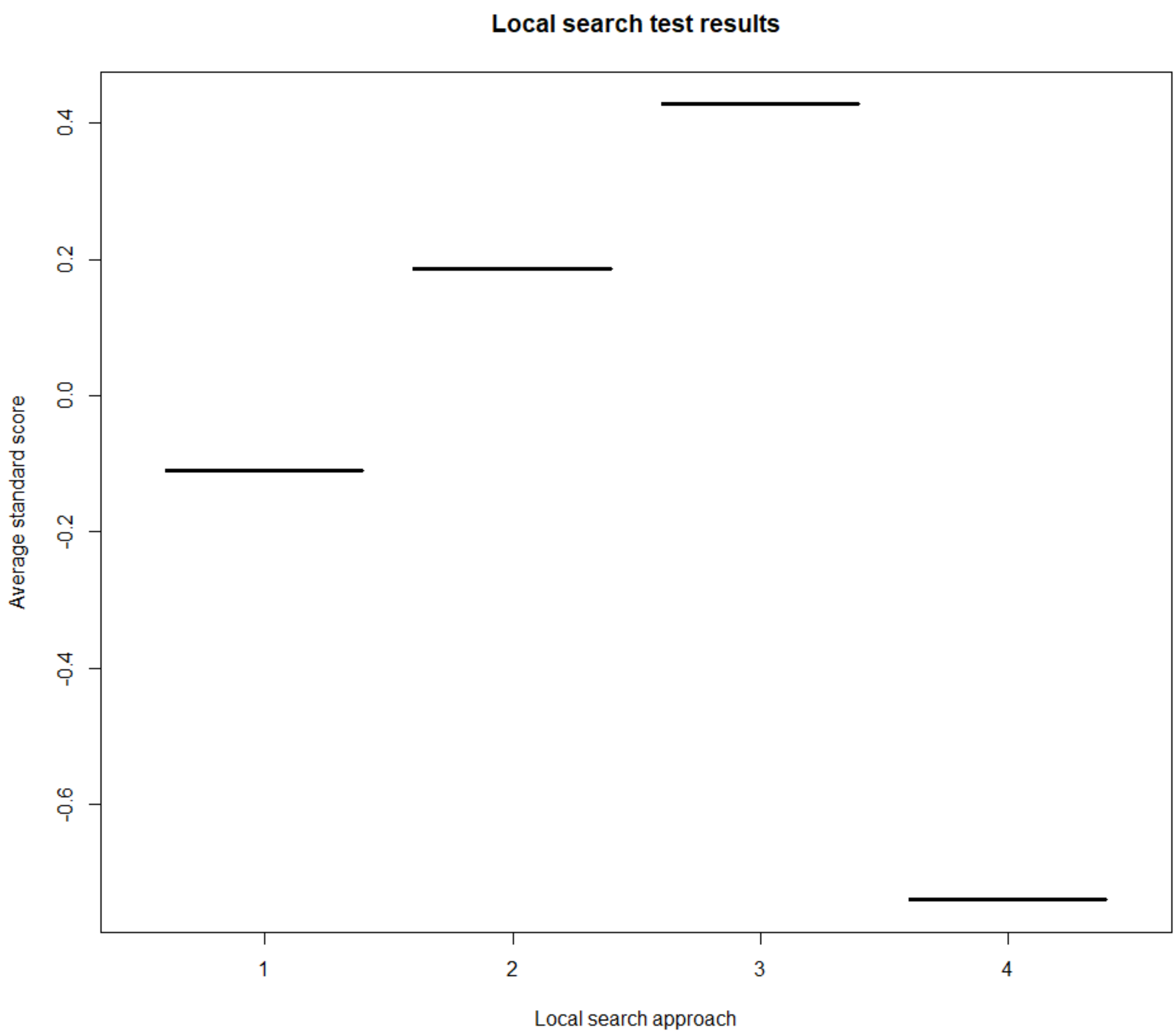

Figure 5.4 Plot of average standard scores for local search testing

Not using local search shows the worst result. Using local search always is better than using it only once. Skipping the local search for some «bad» solution improves the result by alowing to constuct more solutions comparing to always running the local search.

Approach 3 (running local search for good solutions) is selected as a part of the final algorithm.

5.4 Final algorithm results

The test set described previously is used to test different algorithms. The algorithms are the following:

- GRASP: GRASP algorithm described in the previous chapter with the parameters tuned in this chapter
- VNS: a variable neighborhood search algorithm starting from a random solution described and implemented by Bentsen and Hvattum 2020
- GRASP + VNS: VNS algorithm which uses GRASP to create an initial solution instead of a random construction
- Greedy + VNS: VNS algorithm which uses the greedy construction described in this thesis to create an initial solution instead of a random construction
- GRASP2 + VNS: the same as GRASP + VNS but with slightly modified parameters of GRASP to make the construction slightly more random ($\alpha = 0.1, \Delta infeasibility = 0.5$)
- CPLEX: (Cplex 2009)
- Local Solver: (Benoist et al. 2011), https://www.localsolver.com/

The runtime for all the algorithms was set to 5 minutes. Local Solver and CPLEX were tested on a different machine, so the running times are not comparable.

For the algorithms combining GRASP and VNS the runtime of GRASP part was limited by 40 seconds for one construction.

The results are presented in the following table:

Table 5.16 Results on the test set

Instance	GRASP	VNS	GRASP + VNS	GRASP2 + VNS	Greedy + VNS	CPLEX	Local Solver
g15	778	2758	2031	2145	1985	2922	2717
g25	5778		5833	9321	100	10952	10849
g35	1867		2550	4438	1527	6764	6760
g45	2996		3532	4750	2301	5428	5493
g5	9069		9332	9202	763	9906	9953
g50	476	4550	2069	2745	2177	5880	5814
g54	1140	3476	2525	2775	2487	3426	3387
sg3dl053000	-85	88	82	56	58	106	102

sg3dl105000	-1389	546	402	426	474	656	642
sg3dl1410000	-3830	1239	-89	390	362	1264	1276
sg3dl144000	-3822	962	-23	416	238	1150	1434
toursg3-15	-397448706		-15210101	5016140	-55832572	172938456	196006617
toursg3-8	-56516747	21670541	6361821	18584323	6601816	40402223	35163004
tourspm3-15-50	-4719		-1039	-422	-619	1370	1464
tourspm3-8-50	-541	300	180	168	182	374	364
100-100-25-1	57018	57194	57018	57092	57194	57194	57194
100-100-q-1	85632	85545	85632	85491	85652	85652	82664
100-10-5-1-0	9590	10018	9929	9917	10000	10018	10018
100-30-15-1-10	16611	18200	18028	18306	18713	17797	18250
100-30-30-0-1		9494	9385	9612	9785		8933
100-50-10-1	38130	38130	38130	38130	38130	38130	38130
100-50-q-1	43866	43917	43917	43917	43917	43917	43917
100-5-5-1-0	9975	10263	10041	10100	10263	10263	10263
250-10-1-0-14	172782	172232	172978	172534	172717	173386	173443
250-30-30-0-0	32394	33990	34063	34128	34161	34323	32511
250-5-2-0-0	76784	78100	77970	77992	78105	78486	78486
500-10-10-1-0	50275	50385	49666	50372	49638	52741	52381
500-30-15-1-0	46746	47517	47096	48532	48136	50404	47350
500-30-30-0-0	81912	81984	82998	82444	82565	82265	80000
500-5-5-0-14	309431	311055	311642	311370	311557	312069	311989
I11	49896	71785	71728	71852	71833	73773	73749
I13	65127	95901	95767	95718	95707	98434	98389
I5	39057	39057	39057	39057	39057	39057	39057
I9	35794	47939	47858	47964	48029	49174	49175
INST01	10301	10334	10369	10400	10395	10714	10706
INST03	10513	10650	10574	10591	10652	10942	10934
INST07	15806	15978	16005	15975	16000	16411	16440
INST18	33517	58538	58110	58384	58448	60461	60458

INST20	41798	72733	72341	72510	72701	75610	75606
INST21	76240	85776	85638	85628	85830	87616	87552
INST24					41080	41892	41098
INST28	109066	131632	131564	131968	132050	134630	134598
RTI09	78062	78062	78062	78062	78062	78062	78062
RTI12	11074	11438	11380	11344	11396	11632	11632
RTI13	101508	104862	104132	103474	104204	105612	105612
lmhn1000m5000num1	676851	704840	717860	713409	713430	691461	727020
lmhn1500m7500num1	1493487	1579112	1605637	1614096	1614169	1530720	1628910
qn1000m10000t2s0c0num0	121552	134184	132924	135985	134342	137996	139032
qn500m2500t2s0c0num0	47572	51791	52138	51819	51478	52816	52467
qn500m5000t2s0c0num0	31996	35684	34880	34583	34890	35607	35173
rn200m1000t10s0c0num0	19340	19540	19481	19509	19540	19499	19492
rn200m1000t10s0c25num4	19777	20614	20423	20580	20614	20510	20570
rn200m1000t40s20c0num0	21983	22076	22063	22034	22076	22072	22030
rn500m1000t100s50c0num0	145684	146683	146591	146540	146683	146649	146582
rn500m1000t100s50c25num0	145572	146533	146455	146473	146533	146533	146417
rn500m1000t25s0c0num4	144313	147906	147388	147837	147952	147860	147320
rn500m1000t25s0c25num4	144020	146262	145708	145978	146262	146086	145943
rn500m1000t25s0c50num0	141608	143372	142972	143123	143372	143113	143217
rn500m2500t25s0c25num4	140773	144931	144817	144766	145068	144654	144700
rn500m2500t25s0c50num0	139879	142306	141719	142123	142313	141968	141568

A missing value in a cell means that a feasible solution was not found.

The results show that GRASP algorithm is able to find a feasible solution for almost every instance (it failed to find a feasible solution for 2 instances out of 60). However, the quality of the solutions obtained by GRASP is usually worse than the quality of the solutions obtained by VNS (where VNS is able to find a feasible solution) or by the combined methods.

One of the research questions is whether GRASP can be used to improve other heuristic methods. A comparison of the results of VNS and combined methods can help to answer this question.

GRASP with the parameters obtained during testing combined with VNS is able to find feasible solutions for 59 instances out of 60. VNS starting from a random solution finds a feasible solution only for 53 instances. But when both methods are able to find a feasible solution GRASP combined with VNS shows a better result on 7 instances and a worse result on 41 instances.

Changing parameter in GRASP combined with VNS for constructing slightly more random solutions result in feasible solutions for 59 instances of the training set and the same number of feasible solutions are obtained using the original parameters. The objective function value after changing the parameters is improved for 36 instances and is worsened for 19 ones.

Combining VNS with greedy construction allows to find a feasible solution. Changing random construction to greedy construction for VNS results in finding a feasible solution for 7 more instances and improves the solution in 20 cases and worsens a solution in 20 cases.

VNS combined with greedy construction compared to VNS combined with GRASP finds a feasible solution for 1 more solution and performs better in 36 cases and worse in 19 cases.

5.5 Discussion

The results on the test set show that GRASP is able to find feasible solutions for the majority of instances. However, the quality of the solutions obtained by GRASP is worse than the quality of the solution obtained by other methods tested in the previous section.

GRASP used instead of a random construction in VNS makes the algorithm more reliable (able to find a feasible solution) but obtaining slightly worse solutions. GRASP parameters were tuned for using GRASP without VNS. A change in the parameters to make GRASP more random resulted in the improvement of the results. This means that the results can be improved further after testing the combined algorithm.

Combining greedy construction with VNS is able to find a feasible solution for all the instances and shows a good quality of the solution. Greedy construction can be used to improve VNS making it more reliable and without lose in the quality of the solutions.

6.0 Conclusion

Binary integer problem has a number of real-world applications including applications in logistics. Problem instances usually contain a huge number of constraints and variables which makes it impossible to use exact methods to solve this problem. There are few heuristic solvers targeted on general BIP but they are not focused on the construction of a solution.

Greedy construction heuristic algorithm and GRASP for general BIP were implemented and tested. The approach of dynamically changing the parameter of GRASP between iterations is created and used to allow the algorithm to solve a variety of BIP classes.

GRASP can be used for constructing solution for general BIP. Implemented construction algorithms are able to improve VNS algorithm making it able to find a feasible solution for more instances.

There are several directions for further research. The algorithm combining GRASP or greedy construction and VNS can be studied. Parameter tuning for the combined method may improve the results. Another option is to change the local search used in GRASP for VNS and use a statistical or machine learning method to decide whether or not to run the VNS after a construction iteration.

Another direction for the research is creating a population-based method to solve general BIP. This method can make use of the algorithm created during the work on this thesis.

Reference list

Al-Shihabi, Sameh. 2021. “A Novel Core-Based Optimization Framework for Binary Integer Programs-the Multidemand Multidimesional Knapsack Problem as a Test Problem.” *Operations Research Perspectives* 8: 100182.

Baessler, Felix. 1992. “A Heuristic 0–1 Integer Programming Method.” *Operations-Research-Spektrum* 14 (1): 11–18.

Balas, Egon. 1965. “An Additive Algorithm for Solving Linear Programs with Zero-One Variables.” *Operations Research* 13 (4): 517–46.

Balas, Egon, and Clarence H Martin. 1980. “Pivot and Complement–a Heuristic for 0-1 Programming.” *Management Science* 26 (1): 86–96.

Benoist, Thierry, Bertrand Estellon, Frédéric Gardi, Romain Megel, and Karim Nouioua. 2011. “Localsolver 1. x: A Black-Box Local-Search Solver for 0-1 Programming.” *4or* 9 (3): 299–316.

Bentsen, Håkon, and Lars Magnus Hvattum. 2020. “Variable Neighborhood Search for Binary Integer Programming Problems.”

Bertsimas, Dimitris, Dan A Iancu, and Dmitriy Katz. 2013. “A New Local Search Algorithm for Binary Optimization.” *INFORMS Journal on Computing* 25 (2): 208–21.

Cappanera, Paola, and Marco Trubian. 2005. “A Local-Search-Based Heuristic for the Demand-Constrained Multidimensional Knapsack Problem.” *INFORMS Journal on Computing* 17 (1): 82–98.

Chu, Paul C, and John E Beasley. 1998. “A Genetic Algorithm for the Multidimensional Knapsack Problem.” *Journal of Heuristics* 4 (1): 63–86.

Cplex, I B M ILOG. 2009. “V12. 1: User’s Manual for CPLEX.” *International Business Machines Corporation* 46 (53): 157.

Davoine, Thomas, Peter L Hammer, and Béla Vizvári. 2003. “A Heuristic for Boolean Optimization Problems.” *Journal of Heuristics* 9 (3): 229–47.

Duarte, Abraham, and Rafael Martí. 2007. “Tabu Search and GRASP for the Maximum Diversity Problem.” *European Journal of Operational Research* 178 (1): 71–84.

Feo, Thomas A, and Mauricio G C Resende. 1989. “A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem.” *Operations Research Letters* 8 (2): 67–71.

Festa, Paola, Panos M Pardalos, Mauricio G C Resende, and Celso C Ribeiro. 2002. “Randomized Heuristics for the MAX-CUT Problem.” *Optimization Methods and Software* 17 (6): 1033–58.

Festa, Paola, and Mauricio G C Resende. 2002. “GRASP: An Annotated Bibliography.” In *Essays and Surveys in Metaheuristics*, 325–67. Springer.

———. 2009a. “An Annotated Bibliography of GRASP–Part I: Algorithms.” *International Transactions in Operational Research* 16 (1): 1–24.

———. 2009b. “An Annotated Bibliography of GRASP–Part II: Applications.” *International Transactions in Operational Research* 16 (2): 131–72.

Geoffrion, Arthur M. 1967. “Integer Programming by Implicit Enumeration and Balas’ Method.” *Siam Review* 9 (2): 178–90.

Glover, Fred. 1965. “A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem.” *Operations Research* 13 (6): 879–919.

Gökċen, Hadı, and Erdal Erel. 1998. “Binary Integer Formulation for Mixed-Model Assembly Line Balancing Problem.” *Computers & Industrial Engineering* 34 (2): 451–61.

Gortazar, Francisco, Abraham Duarte, Manuel Laguna, and Rafael Martí. 2010. “Black Box Scatter Search for General Classes of Binary Optimization Problems.” *Computers & Operations Research* 37 (11): 1977–86.

Helmberg, Christoph, and Franz Rendl. 2000. “A Spectral Bundle Method for Semidefinite Programming.” *SIAM Journal on Optimization* 10 (3): 673–96.

Hristakeva, Maya, and Dipti Shrestha. 2004. “Solving the 0-1 Knapsack Problem with Genetic Algorithms.” In *Midwest Instruction and Computing Symposium*.

Hvattum, Lars M, Arne Løkketangen, and Fred Glover. 2005. “New Heuristics and Adaptive Memory Procedures for Boolean Optimization Problems.” In *Integer Programming*, 17–34. CRC Press.

Jeong, Seh-Woong, and Fabio Somenzi. 1993. “A New Algorithm for 0-1 Programming Based on Binary Decision Diagrams.” In *Logic Synthesis and Optimization*, 145–65. Springer.

Kallrath, Julia, Steffen Rebennack, Josef Kallrath, and Rüdiger Kusche. 2014. “Solving Real-World Cutting Stock-Problems in the Paper Industry: Mathematical Approaches, Experience and Challenges.” *European Journal of Operational Research* 238 (1): 374–89.

Khan, Shahadat, Kin F Li, Eric G Manning, and Md Mostofa Akbar. 2002. “Solving the Knapsack Problem for Adaptive Multimedia Systems.” *Stud. Inform. Univ.* 2 (1): 157–78.

Laguna, Manuel, Thomas A Feo, and Hal C Elrod. 1994. “A Greedy Randomized Adaptive Search Procedure for the Two-Partition Problem.” *Operations Research* 42 (4): 677–87.

Lai, Xiangjing, Jin-Kao Hao, and Dong Yue. 2019. “Two-Stage Solution-Based Tabu Search for the Multidemand Multidimensional Knapsack Problem.” *European Journal of Operational*

Research 274 (1): 35–48.

Marinescu, Radu, and Rina Dechter. 2010. “Evaluating the Impact of AND/OR Search on 0-1 Integer Linear Programming.” *Constraints* 15 (1): 29–63.

Resende, Mauricio G C, and Thomas A Feo. 1996. “A GRASP for Satisfiability.” In *CLIQUES, COLORING, AND SATISFIABILITY: THE SECOND DIMACS IMPLEMENTATION CHALLENGE, VOLUME 26 OF DIMACS SERIES ON DISCRETE MATHEMATICS AND THEORETICAL COMPUTER SCIENCE*. Citeseer.

Resende, Mauricio G C, L S Pitsoulis, and P M Pardalos. 1997. “Approximate Solution of Weighted MAX-SAT Problems Using GRASP.” *Satisfiability Problems* 35: 393–405.

Resende, Mauricio G C, and Celso C Ribeiro. 2003. “Greedy Randomized Adaptive Search Procedures.” In *Handbook of Metaheuristics*, 219–49. Springer.

Shojaei, Hamid, Twan Basten, Marc Geilen, and Azadeh Davoodi. 2013. “A Fast and Scalable Multidimensional Multiple-Choice Knapsack Heuristic.” *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 18 (4): 1–32.

Silva, Rodrigo Ferreira da, Lars Magnus Hvattum, and Fred Glover. 2020. “Combining Solutions of the Optimum Satisfiability Problem Using Evolutionary Tunneling.” In *MENDEL*, 26:23–29.

Vianna, Dalessandro Soares, and José Elias Claudio Arroyo. 2004. “A GRASP Algorithm for the Multi-Objective Knapsack Problem.” In *XXIV International Conference of the Chilean Computer Science Society*, 69–75. IEEE.