universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„GRASP for the two-dimensional Strip packing problem“

verfasst von / submitted by

Daniel Rovenský, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2020 / Vienna 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 915
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Betriebswirtschaft
Betreut von / Supervisor:	O.Univ.Prof. Dipl.-Ing. Dr. Richard F. Hartl
Mitbetreut von / Co-Supervisor:	Mag. Dr. Andrea Seidl

Abstract

The strip packing problem represents an NP-hard problem with applications in various industries. This thesis is exploring the employment of different simple heuristics like the Bottom-Left heuristic or Touching Perimeter heuristic inside of a more complex GRASP metaheuristic setting. The best performing heuristic is chosen for every phase of the GRASP metaheuristic. The implemented and optimized GRASP metaheuristic is tested on the instances most commonly used in literature and compared to some of the best and newest metaheuristic algorithms used for solving this type of problems. In conclusion the implemented metaheuristic performs reasonably well considering its simplicity and represents a viable option for solving various strip packing problems.

Abstract in German

Stip-Packing-Propleme repräsentieren einen Typ von NP-harten Problemen, welche in den verschiedensten Industrien auftreten. In dieser Masterarbeit wird der Einsatz von verschiedenen einfachen Heuristiken, wie Bottom-Left oder Touching Perimeter, für eine komplexere GRASP Metaheuristik untersucht. In jeder Phase der GRASP Metaheuristik werden dann die Heuristiken mit der besten Performance genutzt. Nach der Implementierung und Optimierung wird die GRASP Metaheuristik an den meistgenutzten Instanzen aus der Literatur getestet und mit den besten und neuesten Metaheuristiken verglichen. Als Fazit kann man feststellen, dass die implementierte GRASP Metaheuristik erreicht unter Berücksichtigung seiner Simplizität relativ gute Ergebnisse und stellt eine brauchbare Alternative zur Lösung der Strip-Packing-Problemen dar.

Contents

List of Figures

List of Tables

1 Introduction

Two-dimensional packing problems have been extensively studied in the literature over many years. In this thesis a GRASP metaheuristic is implemented, rigorously tested on majority of test instances from the literature and compared to the results of many other metaheuristics used for solving this type of problems. This thesis provides an introduction to two-dimensional packing problems, but mainly deep knowledge about the GRASP metaheuristics and specifically the maximal rectangles implementation based on work on Jylänki (2010), which can be used for solving packing problems. The results obtained show potential for the use of simple GRASP like implementations. This is because not only the fully implemented GRASP metaheuristic, but even its simplified version yielded good results in comparison to more sophisticated approaches used in the literature. This simplified version consisted only of the randomised iterative use of construction part of GRASP metaheuristic. The fact that the simplified version yielded comparable results to the GRASP metaheuristic itself, leads to conclusion that randomisation of the construction heuristic represented the most crucial part of the GRASP metaheuristic and the local search phase only led to small improvements. Naturally, this conclusion may be true only for the heuristics used in this thesis or only packing problems, as Feo & Resende (1995) advocate in their work the importance of local search in other GRASP implementations. This encourages further research, exploring the simplified version more in detail and possibly introducing further improvements, like parallel computing, to the implementation.

The classification of two-dimensional packing problems, which can be found for example in the works of Lodi, Martello, & Vigo (1999) and Lodi, Martello, & Vigo (2002) describes four different types of two-dimensional packing problems according to the orientation of the items and the guillotine cutting constraint. This thesis concentrates on the most common variant in the literature, where the orientation of the items is fixed, so these cannot be rotated in any way, and no guillotine cutting constraint needs to be uphold. Naturally, the packing problems can be further divided to either bin or strip packing problem. This further diversification is based on the nature of the container used for the packing.

Strip-packing problem involves the placement of rectangular items onto a strip with a fixed width and infinite height. This finds its application in various industries. An example of its use could be cutting finite number of items from a standardized roll of material. The objective of the strip-packing

problem, in this sample setting, would be to cut all the items from the roll, while minimizing the length of the roll used and therefore minimizing the waste.

Two-dimensional packing problems belong to the strongly NP-hard problems, as can be found in literature in the work of Lodi et al. (1999), Hopper (2000) and many others. This implies that the heuristic approaches are frequently used for solving of these problems and Hopper (2000) further explains that achieving acceptable computation times is the reason for this. Heuristic algorithms naturally evolved through the time. It started with the simple heuristics like the Bottom-Left implemented in the work of Chazelle (1983) and lead to the iterative methods used today delivering results close to the optimum.

This thesis is utilizing a GRASP metaheuristic in order to solve the strip-packing problem. A reactive GRASP metaheuristic was already applied in the work of Alvarez-Valdes, Parreño, & Tamarit (2008) and was regarded in the subsequent and even quite recent works by Zhang, Che, Ye, Si, & Leung (2016), Leung, Zhang, & Mong Sim (2011) and many others as an excellent algorithm for solving of the strip-packing problems. In contrast to the work of Alvarez-Valdes et al. (2008) the GRASP metaheuristic implemented in this work is constructed using different heuristics during the construction phase and introduces active search for holes in the pattern for the local search phase. In addition, this implementation doesn't apply the self-adjustment of the parameters of the GRASP metaheuristic as introduced in the work of Prais & Ribeiro (2000).

The goal of this thesis is to implement a GRASP metaheuristic for the strip-packing problem in a novel way and compare the results with the work of Alvarez-Valdes et al. (2008), other well performing recent papers and an iterative randomized greedy heuristic. The last algorithm is a simplification of the GRASP metaheuristic implemented in this thesis.

The following chapters of this thesis are organized in following way. First, the GRASP metaheuristic is explained briefly. Next, the heuristics in each of the building blocks of the implemented GRASP are explained in more detail. Then, the implementation of all the heuristics and the final metaheuristic is detailed. After that, the preliminary testing and setup of parameters is described, and the last chapter compares and concludes the computation results.

2 Greedy Randomized Adaptive Search Procedures

As already mentioned in the previous chapter the strip-packing problem represents a difficult combinational problem. According to Feo & Resende (1995) GRASP metaheuristic is one of the most promising techniques used for solving of these types of problems, together with techniques like Simulated annealing, Tabu search and Genetic algorithm. They highlight adaptability and the ability to avoid getting stuck in local optima as key to success for all these techniques. Naturally, each of these algorithms uses a different approach in attaining these key features.

GRASP which stands for Greedy Randomized Adaptive Search Procedures is described in works of Resende & Ribeiro (2003) and Feo & Resende (1995) as iterative metaheuristic composed of construction phase and local search phase. Feo & Resende (1995) further describe GRASP as a randomized sampling technique and each of iteration as a sample solution, where in a perfect scenario each of these solutions represent a random local optimum. The ambition of this iterative process is that one of these local optima will represent the global optimum.

The Figure 1 represents a generic flowchart of the GRASP metaheuristic according to Feo & Resende (1995). The authors explain that in each iteration first, a feasible solution is constructed with the use of a greedy randomized heuristic. Next, the neighbourhood of the solution is investigated for local optima. Then, if the achieved solution is better than the current best solution, it will be saved, and the next iteration can start.

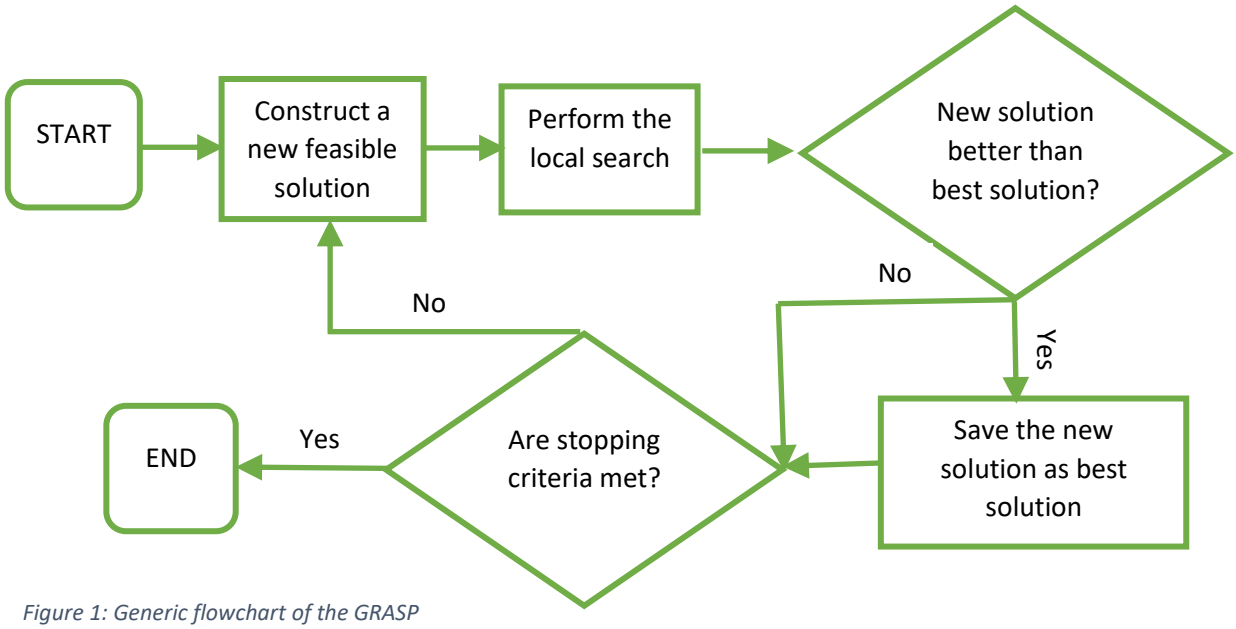

Figure 1: Generic flowchart of the GRASP

Naturally, Feo & Resende (1995) further mention that once any ending criterion is met, the algorithm is stopped, and the present best solution represents the result of the GRASP metaheuristic. They list the possible stopping criteria as the reaching of a set number of iterations or reaching of the optimal solution. In addition to these listed stopping criteria, the current metaheuristic-based implementations for strip-packing widely use the reaching of a time limit as an additional stopping criterion. The reaching of the time limit together with the maximal number of iterations will be used as stopping criteria in this thesis.

The simplicity of the idea behind the GRASP metaheuristic and the ease of its implementation are the main advantages mentioned by the authors Feo & Resende (1995). The authors Feo & Resende (1995) further highlight the ease of the parallel processor implementation as a possible extension of the GRASP metaheuristic, which can greatly improve its performance. They demonstrate this improved performance on an example from work of Feo, Resende, & Smith (1994), where an implementation of the metaheuristic with the use of 8 processors yielded an average improvement of 93%.

The next sections of this chapter concentrate on the construction phase and local search phase of the GRASP metaheuristic and explain each in detail.

2.1 Construction phase

According to Feo & Resende (1995) the aim of the construction phase is to build a feasible solution by applying simple greedy randomised algorithm and preceding one element at a time. This section of the thesis is going to discuss the concept of the greedy nature of the algorithms and their randomisation, while in the next chapter the exact heuristics used will be discussed in more detail.

As already mentioned, a greedy algorithm is used in order to build a feasible solution in construction phase (Feo & Resende, 1995). The simplified premise of the greedy algorithms derived from the work of Bang-Jensen, Gutin, & Yeo (2004) is to always use the most promising candidate first, which is in packing problems the most promising not yet packed item. The most promising candidate is practically an item exhibiting an attribute that makes it advantageous when packed before the other items. This problem can be visualized with a simple example of how a glass jar can be filled with rocks and sand. If the glass jar would be filled with sand first and after that with the rocks the jar will be fuller than if the jar would be filled with rocks first and then the sand would be poured inside. This is because the sand can still fill in the spaces between the rocks, which doesn't work the other way around. The

attribute in this small example is the size of the rocks and it is more promising to put rocks first than the sand, which essentially represents very small rocks. This means that in order to identify the most promising candidate at each point during the packing each item has to be ordered in accordance to a greedy function representing the myopic benefit as presented by Feo & Resende (1995). The function representing the myopic benefit in packing problems can be defined in various ways and Jylänki (2010) states that the ordering of the elements is usually done according to their width, height, area, perimeter and other different criteria. His results further show that not the same sorting criterion is advantageous in all cases. For this reason, a combination of parameters width, height and area with different weight assigned to each is used as the deciding attributes. By simply putting weight on each criterion the manipulation of the ordering is simplified, allowing for aimed changes during the testing phase and investigating their effect on the results. During the testing phase several weight combinations are examined and based on the results the best is chosen for the final implementation.

At this point it is important to note that Feo & Resende (1995) furthermore mention, that the candidate list needs to be updated after each single element was added to the solution. They point out that the reason for this is the effect selecting one element has on the score of the greedy function for the remaining elements. Feo & Resende (1995) clarify that this process of updating the candidate list is what makes the heuristic adaptive. Since the construction heuristics used in this GRASP use width, height and area as sorting attributes, which don't change during the packing itself, this implementation does not contain this adaptive aspect and updating the candidate list after each item is packed would not lead to any change in order of the candidate list.

Another important tool which affects the results greatly is the extend of randomisation of the greedy algorithm used. Feo & Resende (1995) describe the randomisation as randomly choosing one of the most promising candidates from the sorted candidate list. They call the list of most promising candidates a restricted candidate list and regulate the scale of randomization by decreasing or increasing the size of this restricted candidate list. The restricted candidate list is quite simply the list containing a certain number of the items with the highest rank after the sorting is done. When the greedy algorithm is used without the randomisation, which means that the restricted candidate list will at any point contain only one the most promising candidate, a feasible solution will be found, but this solution will always be the same. Feo & Resende (1995) mention that this solution will in many cases be good but suboptimal. They further state and show on an example that with the amount of randomisation variation of the solutions changes and larger randomisation leads to poorer average solution, but ultimately to a better best solution.

The amount of randomisation is important, because solutions are often log-normally distributed. This means that changes in the randomization can greatly affect the quality of the solution and solving time. The reason for this is that because of the log-normal distribution the number of inferior results grows quicker than the number of superior results. It follows from this that when the randomisation is too small, better solutions may never be found, but too much of randomization leads to many more inferior solution and thus longer computation times.

The randomization in thesis will be adjusted through choosing with a certain probability the first, second, third or sometimes even the fourth most promising candidate from the sorted candidate list. Each of these candidates will be chosen with a set probability and several different probability distributions will be tested. In this way the most suitable setting for these problems will be explored, because according to Feo & Resende (1995) improvements of the initial solution greatly improve the efficiency of subsequent local search and consequently even the eventual final solution.

2.2 Local search phase

This phase of the GRASP is aimed at improving the already constructed solution. Multiple heuristics like large neighbourhood search and hill climbing can be applied for this purpose. At this point of the thesis only a general description of the local search phase is presented. The next chapter will then again discuss the most used local search heuristics and justify why large neighbourhood search is used in this thesis.

Feo & Resende (1995) describe in their work the local search as an iterative algorithm improving the current solution by replacing this with a better one. They clarify that the new better solutions are chosen by investigating the neighbourhood of the current solution and that the local search is stopped once no better solution is found after multiple attempts of finding one. The authors further state three reasons why the combination of randomized greedy and local search is beneficial. First, the solutions obtained by the randomized greedy algorithm in the construction phase are often not locally optimal and improving these solutions is advantageous much of the time. Second, the efficiency of local search procedures improves greatly with an excellent initial solution in use. This improved efficiency often results in GRASP generating results quicker and in higher quality than an alone standing local search procedure starting from a random starting point would. Third, local search procedures often end up being stuck in a local optimum and the randomized sampling nature of GRASP helps to overcome this problem by terminating the pointless local search in one neighbourhood and jumping to another neighbourhood of solutions which may turn out to yield better results.

3 Heuristics

This chapter will concentrate on the heuristics used in this thesis, starting with Bottom-Left-Fill, Adjusted Touching Perimeter and Hybrid Heuristic, used during the construction phase, and continuing with the Large Neighbourhood Search (LNS), utilized during the local search phase. Every section will focus on different one of these heuristics, explaining how they work and how do they differ from each other. The final GRASP metaheuristic will be implemented with one of these heuristics for construction phase and one for the local search. The choice of the heuristic utilized for the construction phase will be decided through preliminary testing, to which a separate chapter at the end of the thesis is dedicated. The Section about the LNS will include information about the hill climbing heuristic. The reason for this is to elaborate on the advantages LNS is holding over the hill climbing heuristic and thus explaining why LNS was chosen for the local search phase of this GRASP implementation.

3.1 Bottom-Left-Fill

The simplest of the heuristics implemented in this thesis is the Bottom-Left-Fill (BLF) heuristic. As briefly mentioned in the work of Hopper & Turton (1999) BLF heuristic works by placing each item at the lowest available big enough location and then as far left as possible. This means that the rule of the lowest possible positions will always be considered first and only if there are multiple positions situated as far down as possible then the leftmost of these positions will be chosen. These authors describe the BLF heuristic as development of the Bottom-Left (BL) heuristic. The main advantage of BLF heuristic is its capability of filling holes in the pattern of already packed items and this is further illustrated on an example in the next paragraph.

Figure 2 and Figure 3, depicted on the next page, represent the placement of rectangles inside of a bin according to BL and BLF heuristics and are an example of a case when these two heuristics behave differently. The pink rectangle is the last item that has been placed inside the bin. As presented in Jakobs (1996) when the BL heuristic is used the items are moved into their position is a sliding Tetris-like manner as far down as possible and as far left as possible. Jylänki (2010) called in his work the BL heuristic a Tetris algorithm based on the way it operates. Jylänki (2010) describes the BLF heuristic in a very pragmatic way as placing the rectangle in a suitable position with lowest x- and y-coordinates.

The results from testing done by this author further show an obvious improvement in results when the BL and BLF heuristics are compared.

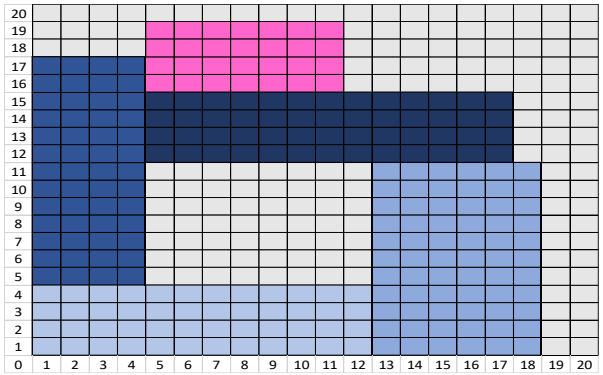

Figure 2: Placement Bottom-Left

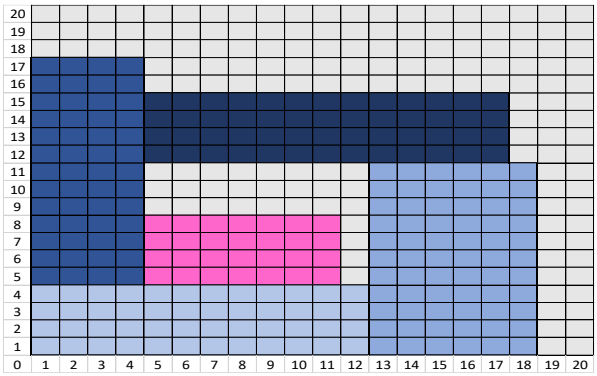

Figure 3: Placement Bottom-Left-Fill

In his work Jylänki (2010) explains that these two mentioned heuristics require a different type of implementation. Because of this, he uses two algorithms Maximum Rectangles Algorithm and Skyline Algorithm. These two algorithms deviate in the way they deal with mapping of the available space. Jylänki (2010) notes, that Maximum Rectangles Algorithm is mapping all the spaces available for packing, while the Skyline Algorithm only keeps track of the topmost edges of the packed rectangles. He further mentions that the BL heuristics can be implemented as the computationally less difficult Skyline Algorithm while the BLF heuristics must be implemented as Maximal Rectangles Algorithm. Since this thesis is using an implementation based on the Maximal Rectangles Algorithm from Jylänki (2010), the better performing Bottom-Left-Fill algorithm was chosen for the implementation.

3.2 Adjusted Touching Perimeter

The next heuristic used is the Adjusted Touching Perimeter heuristic. As the name suggests, this heuristic is based on the Touching Perimeter heuristic. For this reason, first the Touching Perimeter heuristic, as described in the literature, will be presented and next, their differences and the Adjusted Touching Perimeter heuristic will be detailed.

Based on the work of Lodi, Martello, & Vigo (1999) the Touching Perimeter heuristic bases the decision of where to place the item according to the length of the perimeter touching with the container or other items already packed inside of the container. They note that in case two or more positions inside of the container have the same length of the touching perimeter the decision of where to place the box is made according to the Bottom-Left heuristic. These authors even further detail that the touching score is not calculated for each potential placement position. In their work, but similarly

in work of Jylänki (2010) the amount of the considered potential placement positions is limited to only the bottom-left stable positions. Bottom-left stable positions are described by Chazelle (1983) as all the positions from where the item cannot be moved any further down or left. This limitation in moving the rectangles left or down comes either because of the wall of the strip, or because of another previously packed item situated in the way of the movement of the rectangle.

The Figure 4 illustrates with pink rectangles all the bottom-left stable placement positions as described in the works of Lodi, Martello, & Vigo (1999) and Jylänki (2010). These rectangles are always placed in a left corner and are always in touch with another object or the wall of the container on their bottom and left side. Thanks to this placement it is impossible to move these rectangles any more left or down inside the container in which these were packed.

The Adjusted Touching Perimeter heuristic used in this thesis works exactly like the usual Touching Perimeter heuristic but considers more potential placement positions in hope of producing better quality packing patterns. This means, that the bottom-left stable rectangle placement is substituted with a bottom stable rectangle placement. As previously explained, the bottom-left stable rectangle placement represents placement of items inside of the bin in such a way that these cannot be moved down or left and this is because of the wall or other items already packed inside. The bottom stable rectangle placement preserves only the rule that the items cannot be moved down anymore and thus leaves more potential placement positions. As a comparison the Figure 4 is visualizing the potential placement positions used by the Touching Perimeter heuristic and the Figure 5 showcases the potential placement positions used with Adjusted Touching Parameter. Here, it can be seen that the rectangles added in the Figure 5 can be moved further left as there is no wall or other rectangle to the left that would physically restrain this movement. The exact choice of the potential placement positions used in the Adjusted Touching Perimeter is further described in the next chapter.

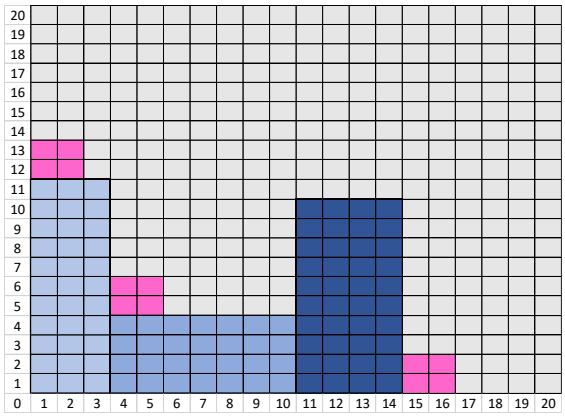

Figure 4: Placement positions Touching Perimeter

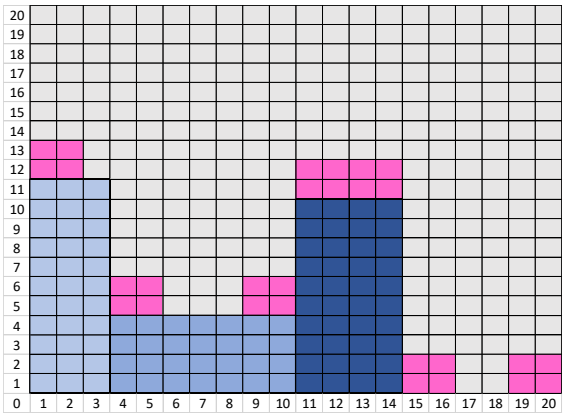

Figure 5: Placement positions Adjusted Touching Perimeter

3.3 Hybrid Heuristic

The last heuristic used for the construction phase is the Hybrid heuristic. This heuristic represents a hybrid between the two Bottom-Left-Fill heuristic and Adjusted Touching Perimeter heuristic. One disadvantage of the Adjusted Touching Perimeter heuristic presented in the previous section is longer computation time. The reason for this long computation time is that the touching score needs to be computed for quite many potential placement positions. It is assumed that especially the placement positions located higher are unnecessary and the Hybrid Heuristic was designed to avoid the computation of the touching score for these placement positions. The goal in building this hybrid heuristic was to create a heuristic that will combine the swiftness of the Bottom-Left-Fill heuristic with the performance of the Adjusted Touching Perimeter heuristic.

To achieve this, the heuristic works in the following way. First, it is determined how far down can the item be packed. This naturally depends on the size of the packed item and the pattern the previously packed items create inside of the container. Next, on this level of deepness all the potential placement positions are identified. After that, the touching score is calculated for these chosen placement positions. Finally, the item is packed on the position with the highest touching score and if two or more positions achieve the same score, then the position positioned to the left is preferred. This hybrid heuristic could also be called Bottom-Touching perimeter as the simple rule of choosing the leftmost position from the bottom left will be replaced with the touching perimeter rule, which means that the position with biggest touching score at the bottom of the rectangle is chosen.

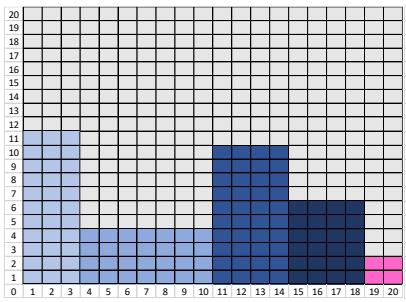

Figure 6: Hybrid heuristic placement 1

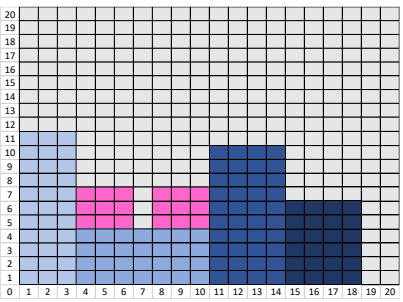

Figure 7: Hybrid heuristic placement 2

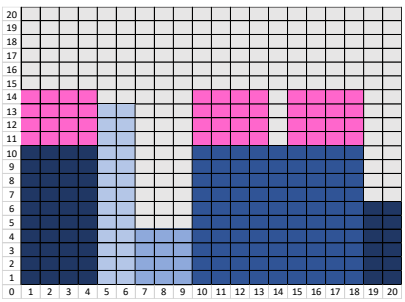

Figure 8: Hybrid heuristic placement 3

The Figures 6 to 8 show how the Hybrid heuristic picks potential placement positions for which the touching score should be calculated. The pink rectangles represent the placements where the item could be placed and the actual shape and size of the packed item. Each of the cases represented in the Figures 6 to 8 is packing different size of rectangles and in this way the figures are portraying different placement positions and possibilities. In the Figure 6 the packed item is small enough to be placed right on the bottom of the container. There is additionally no room for the item to be moved right or left,

so there is only this option on this level. In the next Figure 7 the deepest location where the item can be placed is in between two other previously packed items. Two viable options of where the item can be packed present themselves here. First, located all the way right or second, all the way left. Since both options have identical touching score the item will be packed left. The last Figure 8 represents another special case. Here the packed item is too big to fit in any of the deeper pits and thus the level must be set higher. The potential placements of this item on this level are, either in between the wall and one of the previously packed items or on top of another item in the middle of the pattern. When the item should be placed on top of another item in the middle of the pattern like shown in the Figure 8, the item will be placed either to the left or the right edge of the item underneath. From these three options shown in the Figure 8 the position located most left will be chosen, because it has the biggest touching score.

3.4 Large Neighbourhood Search

Large Neighbourhood Search (LNS) is the heuristic used during the local search phase of the implemented GRASP metaheuristic. In this section of the thesis, first the generic idea behind the LNS will be explained. Later, the details of how the LNS is implemented in the setting of strip-packing problem will be clarified. In addition to the information about the LNS, one more local search procedure called Hill Climbing will be discussed. The reason for this is that Hill Climbing was in the beginning considered for the implementation and it is necessary to explain the reasons behind choosing the LNS instead of this alternative. This means that at this point the advantages of LNS in comparison to the Hill Climbing will be discussed.

In accordance with the work of Pisinger & Ropke (2010) LNS can be simply described as a process of guided destruction and repair of an initial solution with the goal of improving the solution. According to their work various destroy functions, starting from simple random removal, can be employed and many of these methods contain a stochastic element. The destroy function implemented in this thesis employs a guided approach, where part of the solution deemed inadequate is removed, but a certain amount of stochasticity is maintained. As for the repair, Pisinger & Ropke (2010) state that a greedy heuristic can be used to repair the solution and this is going to be the case in this thesis as well. They further show that the range of the searched neighbourhood rests upon the choice of the destroy and repair method used, where destroying larger part of the solution naturally leads to larger neighbourhood being searched.

LNS represents an iterative process as repair and destroy functions are initiated multiple times until an optimal solution is found or no additional improvements are identified after multiple iterations. In order to implement the LNS for the purpose of the strip-packing problem following steps are taken during each of its iteration. In the first step, the holes occurring in the pattern of the initial solution are identified. The reason for this is that especially the bigger holes potentially indicate suboptimality of the previous solution and point to the possible root of this suboptimality. The Figure 9, presented on the next page, represents the initial solution obtained during the construction phase and all the holes occurring are highlighted in pink. In the next step, one of the biggest holes contained in the pattern is chosen and all the items starting at the same level or higher are unpacked. The Figure 10 shows the container after unpacking. In this case, the biggest hole with the area 3x3 was chosen as the unpacking point from all the previously highlighted holes. Then, the partially destroyed solution will be repaired. During this step one of the heuristics used previously during the construction phase is applied and items are packed back into the container. The order in which these items were previously packed inside of the container is during this repacking slightly changed. The change in the packing order is ensuring that a different solution is found after the repacking. Last, if the new solution represents a better solution, this solution replaces the initial solution. This iterative use of destroy and repair procedures is repeated over and over until no improvement is found multiple times in a row.

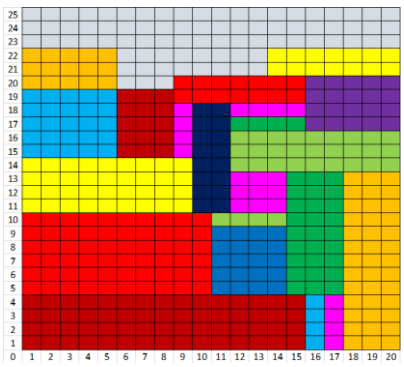

Figure 9: Holes in pattern

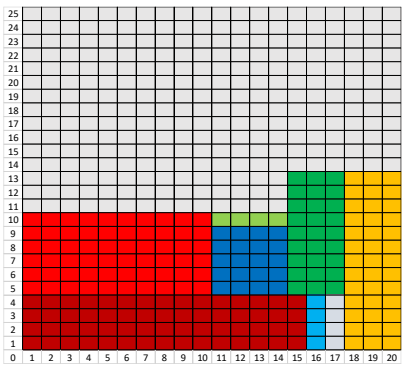

Figure 10: Container after unpacking

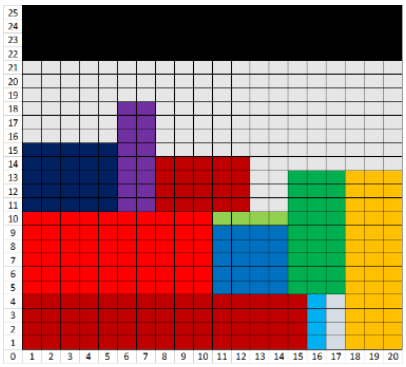

Figure 11: Repair phase

The repair heuristic applied in the LNS is chosen from among the construction heuristics presented in the previous sections and a bin-packing version of these heuristics. This bin-packing version of the previously presented heuristics is specific, because the length of the strip available for the packing of the rectangles is limited. The strip available for packing in this specific case is always a little bit shorter than the height of the previous solution, which means that only better solutions are allowed as the new solution. Figure 11 represents the repair phase in progress, where only limited length of the strip is used for packing. The height of the initial solution as shown in the Figure 9 was 22 and for this reason the maximal height allowed, as shown in the Figure 11, is 21. The reason for

implementing this version of the construction heuristics is that some of these heuristics, mainly the Touching Perimeter heuristic, showcase improved performance in the setting where both width and height of the container are fixed. The improved performance in this setting was discovered in the early phase of implementation and introduction of this bin-packing version to the algorithm further manifests its effectivity during the preliminary testing phase.

At this point the hill climbing will be shortly explained as described in the works of Burke & Kendall (1999) and further Hopper & Turton (2001). These authors describe the hill climbing as an algorithm where the sequence of the candidate list is manipulated. The candidate list in this sense represents an ordered list of all the items, where the first item in the list is packed first and so on. The manipulations according to these authors are in most of the cases random or a partially random swaps of two items in the candidate list. Further they mention that after the swap the solution is constructed using this changed candidate list and if the quality of the new solution is better it replaces the old solution. As an example, after the swap of the 5-th and 10-th item, the 10-th item will be packed 5-th and the 5-th item will be packed 10-th with the use of non-randomised construction heuristic. This means that the hill climbing procedure consists of repetitive use of the same construction heuristic, where the optimal packing order is being investigated by the means of previously mentioned swaps.

The LNS provides two advantages in comparison to the hill climbing heuristic mentioned in the previous paragraph. First, it can be potentially quicker as every time only a part of the solution is destroyed and needs to be reconstructed, while during the hill climbing a whole new feasible solution is constructed every time a change in the candidate list is done. Second, the use of various construction and local search (repair) heuristics can be combined. Combining multiple construction/repair heuristics can lead to improvement in the overall solution, as each heuristic can often be more suitable in different specific problem settings. The assumption that the use of multiple combined heuristics improves the overall solution was verified during the conducted preliminary testing, see Sect. 5.5.

4 The Implementation

In this section the implementation of the GRASP metaheuristic will be described further in detail. The reasons for this are broadening the understanding of how the implemented GRASP metaheuristic and its components work and making further replication studies and improvements possible. The focus of this part of the thesis is not to go in the technicalities of the C++ language, but rather to elaborate on the methodology and problems, which occur during the implementation, and their solutions. After reading this section of the thesis an experienced programmer should be able to replicate our results using C++, Python or any other programming language suitable for the implementation of these heuristics.

The ways of how the packing problems are implemented and solved developed throughout the time and as Chazelle (1983) is mentioning different implementations solve the problem at hand with various efficiencies. Naturally, the choice of implementation must be compatible with the chosen heuristic. In the work of Jylänki (2010) multiple implementations are used in order to solve the packing problem with variety of different heuristic. This thesis is based on the Maximal Rectangles Algorithm presented by Jylänki (2010), which is thanks to its complexity compatible with a multitude of heuristics. The most influential parts of Jylänki's (2010) work was the idea about splitting the free space and updating it every time an additional item was packed.

The rest of this chapter is discussing the implementation in detail. In first sections all the items and containers are defined. Next, the construction heuristics and implicitly the choice of where to pack each item during the packing is examined. Then, the necessary splitting and updating of the free space after each item is packed is discussed. After that, the implementation of the local search procedure is explained and last, the necessary looping of different functions of the implementation will be outlined.

4.1 Definition of the items

To work efficiently with a wide range of different rectangular items which are to be packed inside of a rectangular container it is necessary to define these in a simple way. For this purpose, the information about the item’s width, height, coordinates and ordering parameter is used in this implementation. At the beginning only the width and height of each item is known. The ordering parameter is calculated for each of the items and stored soon after the start of the program. Once the

sorting of the items is finished, they are ready for packing. Coordinates of an item are decided first during packing of this item and determining the right coordinates for each item represents the most crucial part of the packing process. The packing itself can in this setting be defined as the assigning of the coordinates (x and y) to each of the items. After the coordinates are assigned to an item it can be said that this item has been packed inside of the container. Once the packing is finished each of the items will have assigned their distinct coordinates. The coordinates of the items must be assigned in such a way that the items will not be overlapping and all will be contained inside of the defined space of the container in order to represent a feasible solution.

At this point it is to mention that the items and free space rectangles are stored in different data containers (e.g. vector) during the implementation. During this implementation three containers are used. The first container storing the information about all the unpacked items, the second container with all the packed items and last container storing the information about the free space. The next chapter elaborates about the last container storing the information about the free space.

4.2 Definition of the free space

Because of the rectangular shape of the container and the items which are being packed inside of this container the free space is once again defined in rectangles. This goes only naturally and there are only two differences between the items and the free space. First, the rectangles defining the free space don't need any ordering parameter and second the free space rectangles can overlap even for a feasible solution. It will be shown later in this section and follows from the work of Jylänki (2010), that the overlapping of the rectangles is important for the efficient search for the potential packing points.

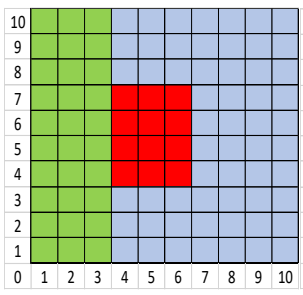

Figure 12: Free space left

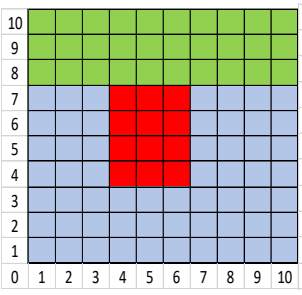

Figure 13: Free space up

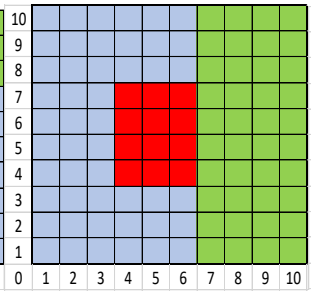

Figure 14: Free space right

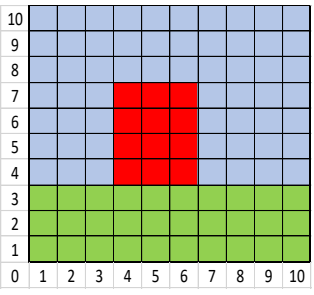

Figure 15: Free space down

The four Figures 12 – 15 above this paragraph illustrate the free space and the overlapping of the rectangles mentioned previously. The red rectangle represents a packed item while the free space is depicted in green and blue. Each of the Figures 12 – 15 is highlighting different free space rectangle,

all of which are inside of one container where only one item (the red rectangle) is packed. Figure 12 is highlighting the left free space rectangle, Figure 13 the free space rectangle above and so on.

The Figures 12 – 15 are illustrating more than only the free space though. The reason why the free space rectangles overlap is that they represent the entirety of the space where an item can be potentially packed and the maximal dimensions of these potential packed rectangles. Because the free space rectangles represent the maximal dimension of the potentially packed rectangles, Jylänki (2010) calls this type of implementation Maximal Rectangles Algorithm. It follows from his work, that if the rectangles would not be overlapping, one would need to consider not only all free space rectangles to define all the possible insertion points, but even all their combinations. This could possibly lead to problems as the search for insertion points will become considerably more complex. Thanks to the overlapping of the rectangles it is sure that, if one item does not fit into any of the free rectangles, it cannot be placed anywhere inside of the space of the container.

As already mentioned, all the information about the free space rectangles is stored in one data container (e.g. vector). In the beginning, there is only one free space rectangle with the coordinates (0,0) and the width and height of the whole container. Since the height of the container is unlimited in strip packing problems the height is simply substituted with a very high number. Every time an item is packed the information about the free space is updated, unnecessary free space rectangles are deleted, and new ones are created as adopted from Jylänki (2010). The details about the updating of the list of the free space rectangles will be presented in the later section of this chapter.

4.3 Packing item inside of the container

As already mentioned in the first section of this chapter, the packing represents the assigning of coordinates to each item inside of the container. Goal of each of the heuristics of the construction phase is to define these coordinates. At the same time the heuristics must assure that the rules of the heuristics are abided, and the feasibility of the solution is guaranteed. In this section the implementation of each of the heuristics is explained in detail and especially the part of how they identify the right coordinates for each of the items.

The Bottom-Left-Fill heuristic could thanks to the nature of the implementation be implemented in a very simple way. The necessary information for identifying the right coordinates are all the information about the next item that is being packed and the updated list of free space rectangles. First, the list of free space rectangles is sorted according to the bottom left criteria. This

means that the free space rectangles with the smallest y coordinates go first and if two rectangles are at the same level, thus have identical y coordinates, the free space rectangle located to the left, with smaller x coordinate, precedes. After the sorting is done, it is checked one free space rectangle at the time, which of these free space rectangles is big enough to fit the item that is being packed. The check is done simply by comparing widths and heights of packed item and free space rectangles. Since the list of the free space rectangles was sorted, the first free space rectangle that is big enough represents the free space rectangle used for packing. The coordinates of this free space rectangle can automatically be assigned to the item that is being packed, because the BLF heuristic is always placing the item in the bottom left corner of the free space rectangle.

The Figures 16 to 18 describe situation before packing an item in the container. The free space rectangles are depicted in grey and the already packed items in red or green. After sorting, the free space rectangles will be in following order. First, is the free space rectangle in the Figure 16. Second, is the one in the Figure 17 and last is the rectangle depicted in the Figure 18. This order will be used to check where an item will be packed inside of this container. If the item's width is shorter than 5 it will be packed to the rectangle in the Figure 16. If it's bigger than 5, but smaller than 10, than the rectangle in the Figure 17 and all the bigger items will be packed to the free space rectangle in the Figure 18.

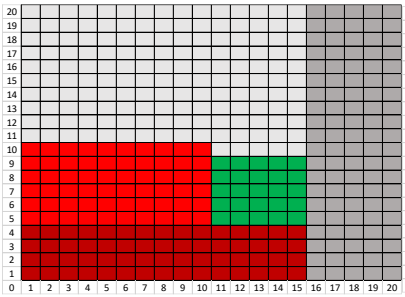

Figure 16: BLF Free space rectangle 1

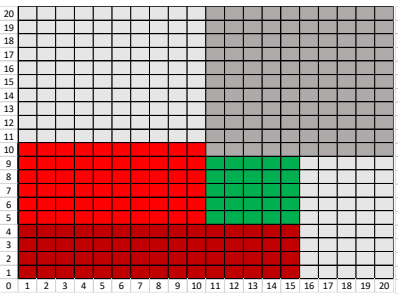

Figure 17: BLF Free space rectangle 2

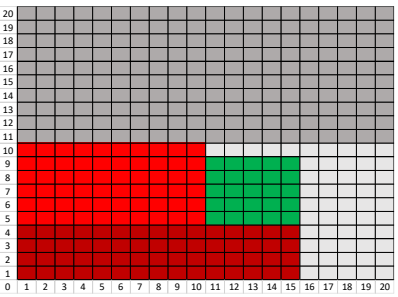

Figure 18: BLF Free space rectangle 3

In comparison with the BLF heuristic the Adjusted Touching Perimeter heuristic is evidently more complicated. To find the coordinate, for the item that is being inserted, it is necessary to calculate the touching perimeter for every potential insertion point. This means that one must search not only all the free space rectangles, but even multiple potential insertion points contained in each of these free space rectangles. This dilemma arises for heuristics that don't respect the bottom-left stable positioning of items as described by Chazelle (1983), and thus for the Adjusted Touching Parameter heuristic and the Hybrid Heuristic used in this thesis.

The Adjusted Touching Perimeter heuristic was implemented in following way. A loop through all the free space rectangles big enough to accommodate the packed item is set up. In this way each of these potentially used rectangles will be searched. In each of these free space rectangles multiple

potential insertion points can be found and all these need to be considered. For each potential insertion point, a touching score needs to be calculated. The potential insertion points considered, are mostly in the lower left or lower right corner of the free space rectangle. These instances, depicted in the Figure 19 are in many cases meaningful, because items packed in these positions are touching the surrounding on two sides simultaneously. In some special cases the potential insertion points can be in the middle the free space rectangle. This case is depicted in the Figure 20. The Figure 20 is highlighting in grey the free space rectangle in which the item could be packed and the two potential packing positions in the middle of the free space rectangle are highlighted in pink. These special cases are meaningful only when pattern inside of container emerges, where placing the item on the side would result in covering a hole creating space which will be hard to fill in later. In the example in Figure 20 the potential insertion point number 1 will be only considered when there is hole nearby just like the one highlighted in red. The insertion point number 2 needs a hole just like the one highlighted in yellow. It can be also seen from this example that positioning the item in the left corner of the grey free space rectangle would result in overall smaller touching perimeter score.

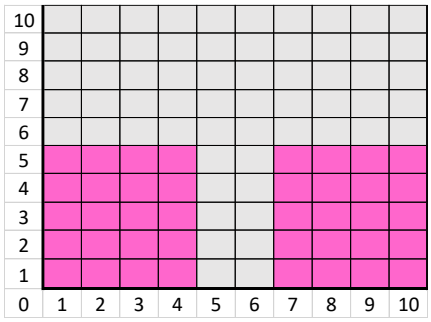

Figure 19: Insertion points in corners

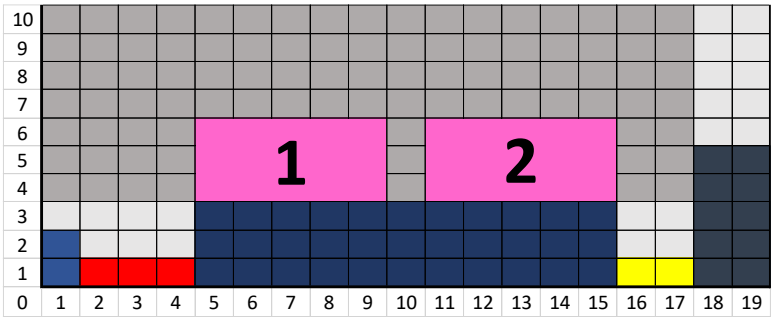

Figure 20: Insertion point in the middle of the free space rectangle

Each time a potential insertion point is discovered the touching score for this point is calculated. This score is calculated by first checking if the item, in the position in question, will be touching any of the walls of the container. In this case the length of the side that is touching the wall will be added to the touching score. Next, it is checked if the item will be touching with any of the already packed items. This will be done by looping through them all. In case they are touching, the length of their contact is calculated, and this is again added to the touching score. After all the already packed items have been looped through the calculation of the touching score should be finished.

Every time a touching score is calculated it is compared to the currently best touching score. In cases when the touching score equals the touching score of the current best insertion position the rule from Bottom-Left heuristic will be used to determine which point will become the new current best. This means that the current touching score and the insertion point affiliated with this will be saved as the current best touching score in these three cases:

- The new touching score is higher
- The new touching score is equal, but the insertion point is located deeper
- The new touching score is equal, and the insertion point is on the same level, but further left

Naturally, after looping through all the free space rectangles is done the coordinates of the insertion point with the highest touching score, as deep as possible and as far left as possible are found.

Now the implementation of the Hybrid Heuristic will be explained. After knowing how the Adjusted Touching Perimeter works it is easy to implement the hybrid heuristic. The key point to this is to use only the free space rectangles located at the bottom, but naturally these free space rectangles need to be big enough to accommodate the packed item. In order to choose only the right free space rectangles this method is used: First, the free space rectangles are sorted according to their y coordinate. Then, the looping through the free space rectangles begins. Once, the first free space rectangle big enough is identified its y coordinate is saved and the touching scores for this free space rectangles are calculated. The touching scores are then calculated for all the following free space rectangles that are big enough to accommodate the item and have the saved y coordinate. Once the turn comes to a free space rectangle with higher y coordinate the loop is broken and no more free space rectangles need to be checked.

As already mentioned, every time a big enough free space rectangle is being looped through, all its potential insertion points are examined and a touching score is calculated for each, compared with the best and saved if necessary. This procedure is the same as with the Adjusted Touching Perimeter. This way, the coordinates of the insertion point in accordance with the Hybrid Heuristic will be found and these can be assigned to the item that is being packed.

Now that the question of where the item should be packed during the use of various construction heuristics is answered, the question of randomisation of the ordered list of items will be outlined. The list of all the items to pack is ordered according to various rules connected to their height, width and area. The most promising rule will be decided during the preliminary testing phase described in detail in chapter five. The randomisation of this order is simply implemented with the use of random number generator. For example, if one of the first three items in order should be used and the first should be used with the chance of 70%, second with the chance of 20%, and third with chance of 10%, then random numbers in the range from 0 to 9 will be generated. If the random generated number is in the range from 0-6 then the first item in order will be taken. If the random generated number is 7 or 8 then the second item in order will be used and if the random number is 9 then the third item in order will be packed. The exact level of randomisation will once again be addressed during the preliminary testing phase, where multiple alternatives will be tested.

4.4 Free space after an item is packed

After packing each of the items it is necessary to consider the changes to the free space inside the container, obviously without this the packing of another item and assuring the feasibility of the solution cannot be guaranteed. There are multiple ways of making these adjustments to the free space and several were considered for this thesis. Eventually, it was decided to use the splitting process proposed in the work of Jylänki (2010). In his work Jylänki (2010) proposes that the free space rectangle is split in such a way that all new free space rectangles stretch in maximal length in every direction. This splitting process is visualized in his work with a figure which has been adapted into the Figure 21. The Figure 21 shows in pink the item that has been placed inside a free space rectangle, then the free space is being split along all of the axes (highlighted in green) and two new free space rectangles emerge. These two new rectangles are overlapping and share the region of space marked with yellow.

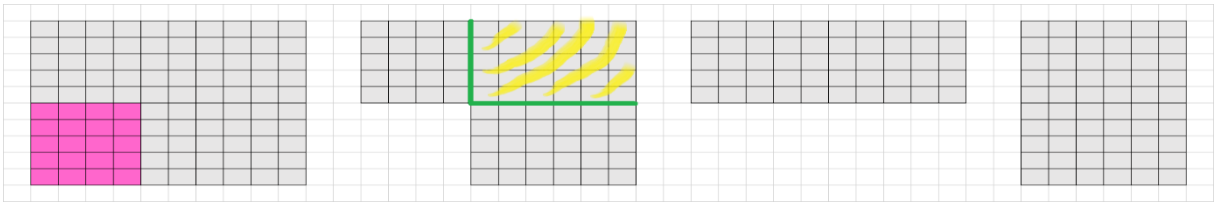

*Figure 21: Splitting process Bottom-Left Adapted from "*A Thousand Ways to Pack the Bin-A Practical Approach to Two-Dimensional Rectangle Bin Packing" by *J. Jylänki, 2010 (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.695.2918&rep=rep1&type=pdf)*

In a case, where the item is not placed in a corner of a free space rectangle, the free space rectangle will be split in three new rectangles. The Figure 22 represents this split into three rectangles analogically to the previous Figure 21.

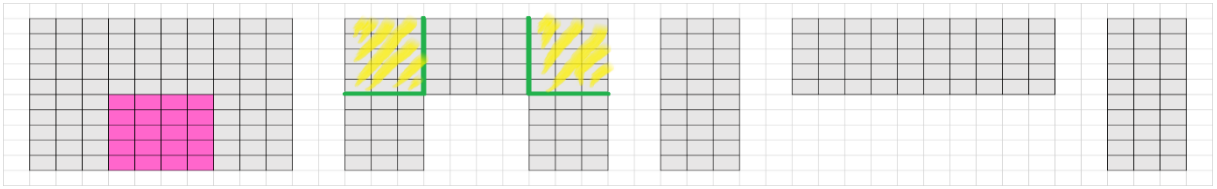

Figure 22: Splitting process In the Middle

According to Jylänki (2010) this splitting process needs to be completed for each free space rectangle the newly inserted item intersects with. He further explains that during the creation of the new free space rectangles some unnecessary rectangles are created as well. For this reason, he proposes that deletion of unnecessary rectangles follows.

The implementation of the splitting itself works in following way. A loop going through all the free space rectangles is started. In case the item intersects with a free space rectangle it needs to be

checked in which way the intersection is happening. Altogether, the intersecting can happen in four positions and combination of these. The four different positions are depicted in the Figure 23.

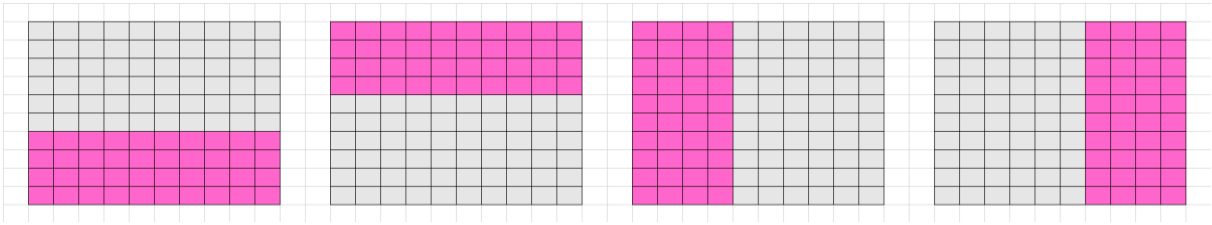

Figure 23: Possible intersections of Item and Free Space Rectangle

The Figure 23 depicts left to right these situations. First, the item is located at the bottom of the free space rectangle and the new free space rectangle will have the y coordinate higher than the original one and the height adjusted accordingly. Second, the item is located at the top, and in this case the new free space rectangle will have smaller height. Third, the item is located left, and the new free space rectangle will have higher x coordinate and adjusted width. Fourth, the item is located at the right side of the free space rectangle and a new free space rectangle will have smaller width.

The four splits explained in the previous paragraph are combined if necessary. The combination of splits results is creation of multiple new free space rectangles, each in accordance with their relative position to the item. For example, in the Figure 21 depicted on the previous page, two new free space rectangles are created. First, with higher y coordinate as the item is located at the bottom and second, with higher x coordinate as the item is on the left side as well. Hence if the coordinates (x, y) of the original free space rectangle were (5, 10), its dimensions (width, height) were (30, 20) and the intersecting part of the item was the size of (10, 10), then the first new free space rectangle would have coordinates (5, 20) and dimensions (30, 10) and the second new free space rectangle would have the coordinates (15, 10) and dimensions (20, 20). In another example, in Figure 22 presented on previous page as well, three new free space rectangles are created. First, according to the rule when the item is at the bottom and next two according to both rules for when the item is left and right, as the item is in the middle of the former free space rectangle. If necessary, all four possible way of splitting can be done simultaneously, creating four new free space rectangles.

Once the splitting of the old free space rectangle is finished this old rectangle must be deleted and after the splitting was finished for all the intersecting rectangles the process of deleting the unnecessary free space rectangles can begin. During this process as described by Jylänki (2010) every single free space rectangle is compared with every other free space rectangle and it is checked if one is located inside of the other one. If this is the case the smaller rectangle is unnecessary and needs to be deleted. The Figure 24, which can be found on the next page, represents a case where the pink item is placed into container, where the only red item was previously packed. There were two free space

rectangles before the splitting. Now they are split in 5 new free space rectangles. Each of the new free space rectangles is highlighted in green in different parts of the Figure 24. The unnecessary free space rectangles are additionally highlighted with yellow. These highlighted rectangles will subsequently be deleted during process of deleting the unnecessary free space rectangles.

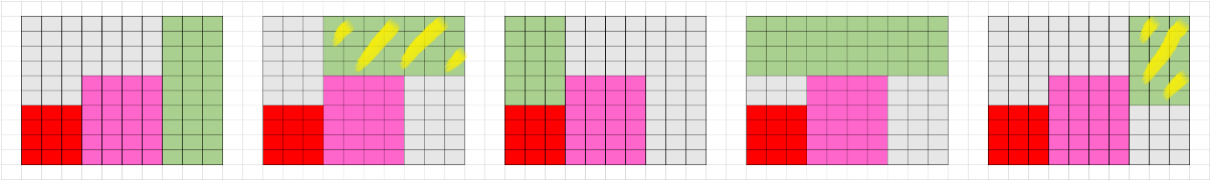

Figure 24: Unnecessary Free Space Rectangles

4.5 Implementing the Local Search Phase

At this point the implementation of the Large Neighbourhood Search is explained in multiple steps. First important step it is to identify the holes in the pattern. These holes in the pattern are represented as the free space rectangles. The only difference between the free space rectangles above the highest items and holes inside of the pattern is that the free space rectangles above the items are always stretching all the way up to the top of the container. Thanks to this property it is easy to differentiate between these two and it is possible to order all the holes in the pattern from biggest to the smallest. Once the holes were ordered it is easy to pick at random one of the biggest from them and in that way decide the level up until which the unpacking will be done.

In the next step in the local search phase the unpacking of the items is carried out. In this step the items from the container with the y coordinate higher or equal to the level, up until which the unpacking is done, are moved back to the container for unpacked items. The items in the unpacked container must stay in the same order as the order in which they were previously packed. The goal of this is that only aimed changes in the order from the original randomized packing order should be done.

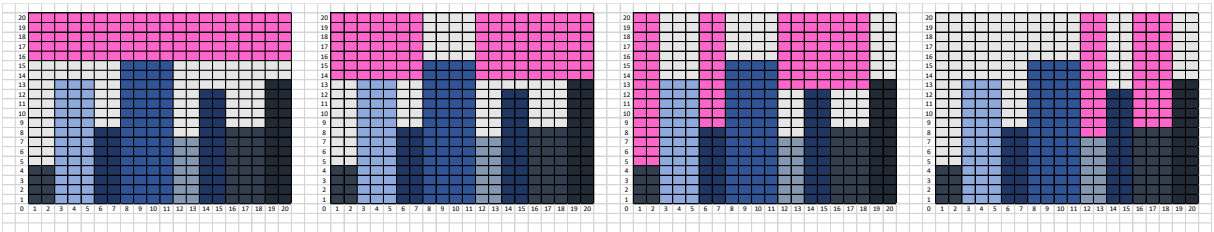

Figure 25: Reconstruction of the Free Space Rectangles

After the unpacking, it is important to reconstruct the free space rectangles inside of the container. This is done by first defining the free space rectangle on top of the highest packed rectangle.

This highest free space rectangle will have the width of the whole container and will stretch all the way up until the top of the container. After this, iteratively the search goes left and right to the lower levels of the pattern of the items packed on top of the container. The width of all the next free space rectangles will be always defined by either wall on the right, bigger item on the right, bigger item on the left or the wall on the left. The Figure 25 above this paragraph shows the algorithm for reconstructing the free space rectangles at work. At each iteration free space rectangles deeper in pattern are reconstructed. At each step the newly construed free space rectangles in the Figure 25 are shown in pink colour. Once all the new free space rectangles were constructed, the unnecessary free space rectangles, left from before the unpacking was done, need to be deleted. This is easily done by running the function for deletion of the unnecessary rectangles as all the unnecessary free space rectangles should lay inside of the newly constructed rectangles.

Once all the previous steps are completed the items unpacked from the container can be packed back to the container by using one of the construction heuristics explained previously. The previous order in which the rectangles were packed inside of the container is slightly adjusted in this step to avoid creating the same patterns multiple times. The adjustments are achieved by randomising the order in which first few rectangles are packed back inside of the container. Multiple ways of randomising the rectangles packed back inside of the container will be explored during the testing phase and the most promising ways will be chosen for the implementation of the GRASP metaheuristic.

At this point of the thesis the transformation of the strip packing problem into a bin packing problem will be discussed. As already mentioned in the previous chapter, this transformation will be used in some of the cases of the repair algorithm of the LNS. Transforming the strip-packing problem into a bin-packing problem can be done simply by adjusting the height of the free space rectangles. This is possible because the container in which the item is being packed during strip packing is already a bin which has its height set as an arbitrarily high number. The free space rectangles which need to be adjusted are all which are stretching all the way until the top of the container. This identification is the exact opposite to identifying the holes in the pattern used in the beginning of this section.

With this all necessary parts of implementing the GRASP Metaheuristic are explained. As in any more complicated algorithm it is highly advisable to implement all the building blocks of the GRASP metaheuristic into separate functions. After this, it is only necessary to loop through the functions in the right way. The way of how all the functions should be looped is best understood by reviewing the flowchart in the Figure 1 found on the page 3, where the GRASP metaheuristic is explained in general.

5 Preliminary testing

The preliminary tests were completed on a limited number of test instances in order to determine the best strategies for the implemented GRASP metaheuristic. The test instances for preliminary testing were chosen arbitrarily from the most used test instances available in the literature. However, it was made sure that test instances from both zero-waste instances and non-zero-waste instances, i.e. instances containing no holes when solved to optimality and instances naturally containing holes will be considered. This chapter is concentrating on presenting the results and implications of the preliminary testing. Nonetheless, the first section is presenting all the test instances used in this thesis and specifies which of these instances will be used for preliminary testing. Further sections then take on the strategies, e.g. construction heuristic, randomisation etc., one after another. The first tested strategies tackled will be related to the construction phase only and the local search phase will be tested in later sections of this chapter.

5.1 Test problems

In this thesis most of the available test instances from the literature are considered, in order to test the implemented GRASP metaheuristic and compare it with other implementations. Broadly, there are two types of test instances. The first type of instances containing no holes in optimality are called zero-waste instances. These instances are usually created by repeatedly slicing an initial rectangle as mentioned by Alvarez-Valdes, Parreño, & Tamarit (2008). The second type of instances are non-zero-waste instances. These instances naturally contain holes even in optimum and Leung, Zhang, & Mong Sim (2011) mention that for some of these instances only a lower bound is used instead of an actual optimal solution when this is unknown.

All the used sets of zero-waste instances are concluded in the Table 1 and the non-zero-waste instances in Table 2. Both tables can be found on the next page. These tables contain the information about the author of the test instances, a denomination under which each set of the test instances is usually found in the literature, number of instances in each set, number of rectangles which needs to be packed, the width of container and the optimal height if known. In addition, the Table 1 and Table 2 mark which of these sets of instances were used during the preliminary testing phase. Naturally, all the mentioned instances were used during the final testing.

Author	Name of Set	Nr. of Instances	Nr. of Rectangles	Width	Height	Preliminary Test
Hopper & Turton (2001)	C	21	16 - 197	20 - 160	15 - 240	Yes
Hopper E. (2000)	N	35	17 - 199	200	200	Yes
Hopper E. (2000)	T	35	17 - 199	200	200	Yes
Burke, Kendall, & Whitwell (2004)	Burke	13	10 - 3152	30 - 640	40 - 960	No
Wang & Valenzela (2001)	Nice	6	25 - 1000	1000	1000	No
Wang & Valenzela (2001)	Path	6	25 - 1000	1000	1000	No

Table 1: Zero-Waste Instances used

Author	Name of Set	Nr. of Instances	Nr. of Rectangles	Width	Height	Preliminary Test
Beasley (1985a)	ngcut	12	7 - 22	10 - 30	-	Yes
Beasley (1985b)	gcut	13	10 - 50	250 - 3000	-	Yes
Christofides & Whitlock (1977)	cgcut	3	16 - 60	10 - 70	-	Yes
Bengtsson (1982)	beng	10	20 - 200	25 - 40	-	Yes
Berkey & Wang (1987)	bwmv	300	20 - 100	10 - 100	-	No
Martello & Vigo (1998)	bwmv	200	20 - 100	10 - 100	-	No

Table 2: Non-Zero-Waste Instances used

5.2 Heuristic for the Construction Phase

The first set of preliminary testing was completed in order to determine, which of the construction heuristics should be used for construction phase. Extensive preliminary testing for all the instances noted in the Table 1 and Table 2 was conducted only for the Bottom-Left-Fill heuristic and the Hybrid heuristic. During these tests several feasible solutions were created for each instance. The testing was interrupted once either 500 feasible solutions were produced, or a time limit of 60 seconds was reached. These stopping criteria were subsequently used in all the preliminary tests related to the construction phase. In this way it is possible to see how the quality of achieved solutions grows, every time another setting is optimized throughout the preliminary testing.

As mentioned previously, from all three proposed construction heuristics only the Bottom-Left-Fill heuristic and Hybrid heuristic are considered for the more extensive preliminary testing. The reason for this was very poor performance of the Adjusted Touching Perimeter heuristic during the construction phase. This poor performance for almost all the instances of strip packing was attributed to the predisposition to stack items on top of each other on the edges of the container. This property of Adjusted Touching Perimeter is helpful in some cases, because it allows to postpone the placing of

an item into an unfavourable position, but it should stop once a critical height has been reached. This means that without the knowledge of any previous solution Adjusted Touching Perimeter will lead to a very poor solutions in strip packing problems and is evidently more suitable for solving bin packing problems.

In order to manifest the disposition of Adjusted Touching Perimeter heuristic for stacking the items on top of each other a series of Figures 26 to 28 is displayed underneath this paragraph. These figures represent a same instance solved with all the construction heuristics. Figure 26 and 27 highlight the different strategies for item placement for each of the heuristics. Figure 28 shows how the packing continues and the stacking of items with Adjusted Touching Perimeter is obvious. In the Figure 28 the Adjusted Touching Perimeter has already exceeded the optimal solution of 20 even though only half of the items are packed at this point.

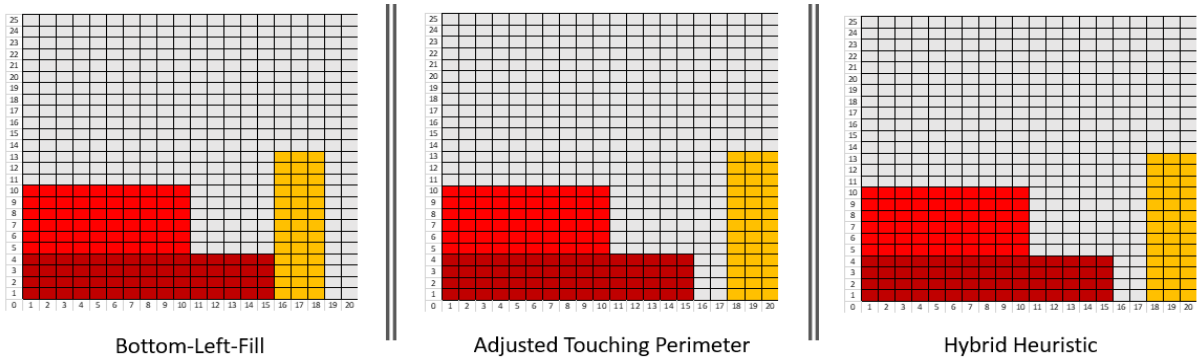

Figure 26: Comparison of construction heuristics 1

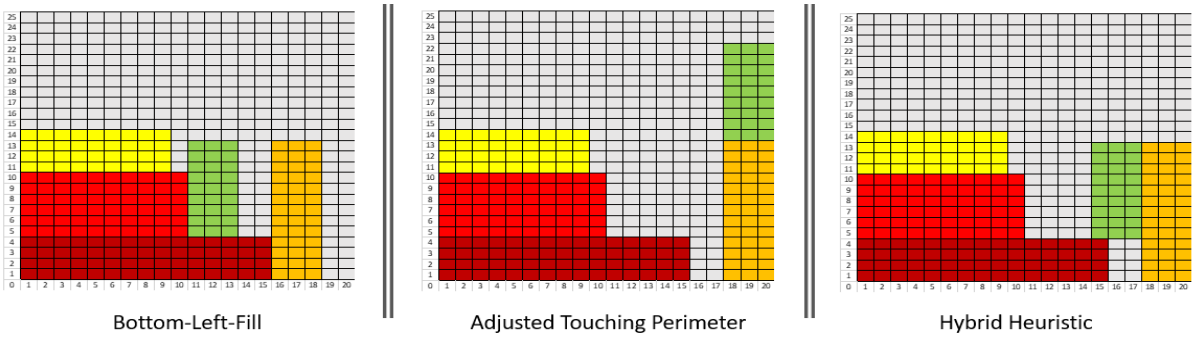

Figure 27: Comparison of construction heuristics 2

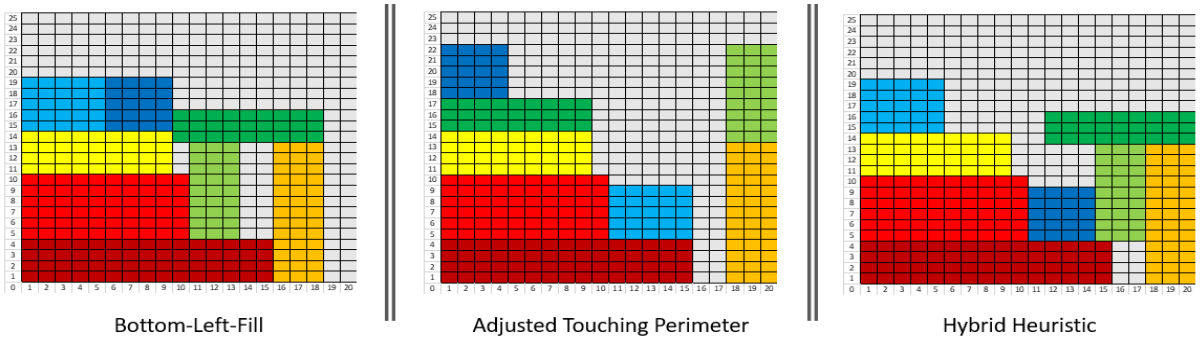

Figure 28: Comparison of construction heuristics 3

The decision between the Bottom-Left-Fill heuristic and Hybrid heuristic was done according to the results of the preliminary testing represented within the Figure 29. Figure 29 represents the average relative distance of each solution found through each iteration from the optimum and the average relative distance of only the best solutions from the optimum. From this graph it is clear, that the Hybrid heuristic outperforms the BLF heuristic in most of the cases. The solutions generated with the Hybrid heuristic are on average 15% better than the solutions from Bottom-Left Fill heuristic and this difference is even slightly higher when only the best solutions from each of the tests are taken into consideration. Based on these results the Hybrid heuristic was chosen for all the subsequent tests.

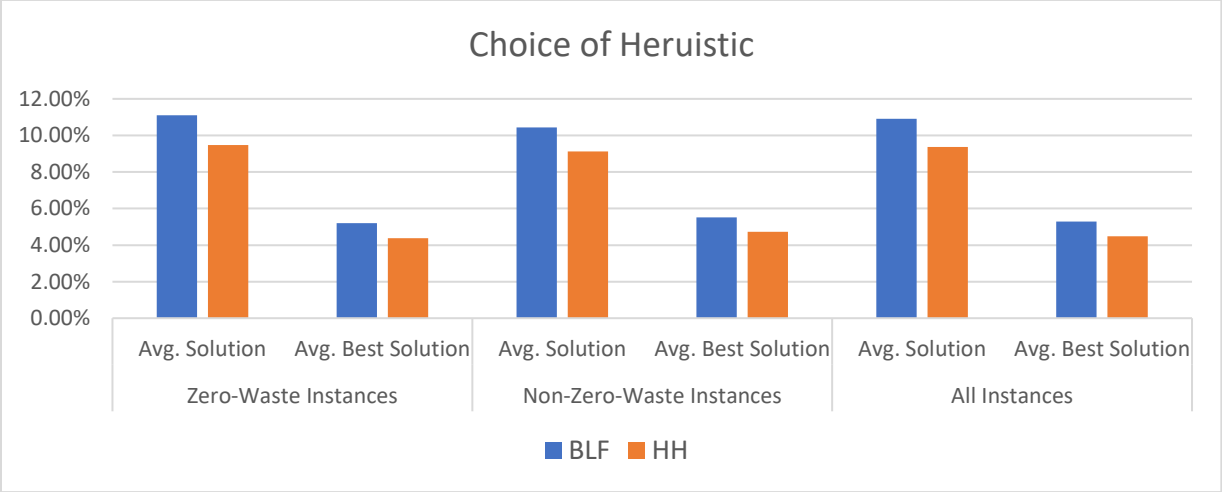

Figure 29: Results preliminary testing choice of heuristic

5.3 Sorting rule of the items

This section will present the results of the preliminary testing conducted in order to identify which sorting rule is the best. As already mentioned, the sorting is done according to the parameters area, width and height, where a weight is assigned to each of these parameters to ease the manipulation of the sorting. Naturally, the parameters height and width are much smaller than the parameter area. This is because area on its own is calculated as multiplication of height and width. In order to avoid this incompatibility of compared parameters, the parameters of height and width will be quadrated in order to be comparable to the area. The next paragraphs will present several weight distributions and their respective performance.

In this paragraph the respective weight distributions used during the preliminary testing will be presented and described as series of three digits. First digit describes the weight assigned to the

parameter representing the area, second width and third the height. The following four tests were conducted at the start of this part of preliminary testing. First a test, where all the weight is assigned to area (1-0-0), thus making it the only sorting parameter. Second test, where the same weight is assigned to each of the parameters (1-1-1), thus making the relative importance of each of the parameters equal. Third test, where the relative importance of width is doubled (1-2-1) and fourth test, where the relative importance of height is doubled (1-1-2). The results showcasing the relative distance from the optimum can be found in the Table 3 underneath of this paragraph. These four tests can be found in Table 3 headlined according to the weights assigned to each parameter as 1-0-0, 1-1-1, 1-2-1 and 1-1-2. The tests 1-1-2 performed best for the zero-waste instances with best solutions averaging at 4,22% point difference from the optimum and the test 1-2-1 for the non-zero-waste instances with the 4,72% difference from the optimum for the best solutions. The other two sorting rules found in some cases the same solutions, but the sorting rules 1-1-2 and 1-2-1 were more consistent in finding a better solution which is showcased with better average result over all the solutions. This demonstrated that in some instances giving more weight for the sorting according to the width and in others instances according to height is favourable to the sorting only according to the area of the rectangles or when the relative importance of the parameters is equal. This initiated further testing.

		1 - 0 - 0	1 - 1 - 1	1 - 2 - 1	1 - 1 - 2	Test	Random 1	Random 2
Zero-Waste Instances	Avg. Solution	9,24%	9,48%	10,69%	8,73%	8,49%	9,48%	9,80%
	Avg. Best Solution	4,22%	4,38%	5,03%	4,22%	4,06%	4,40%	4,09%
Non-Zero-Waste Instances	Avg. Solution	11,01%	9,12%	8,78%	9,59%	8,24%	9,09%	9,24%
	Avg. Best Solution	5,68%	4,73%	4,72%	5,07%	4,40%	4,78%	4,19%
All Instances	Avg. Solution	9,76%	9,37%	10,13%	8,98%	8,42%	9,37%	9,63%
	Avg. Best Solution	4,65%	4,48%	4,94%	4,47%	4,16%	4,51%	4,12%

Table 3: Results preliminary testing choice of sorting rule

The further testing consisted of the following test, which can be found in the Table 3 in the column “Test”. This test was created in order to explore if further increase in relative importance of width or height will lead to better results and used parameters 1-3-1 for instances which previously reached better results in the second test (1-2-1) and the parameters 1-1-3 in those instances performing better during the third test (1-1-2). The results of this test were reaching relative distance from the optimum in best solutions 4,16% on average, which was better than in any of previously performed tests and even the overall variation of the results showed significant improvement. This confirmed that further increase in relative importance of width is beneficial for one group of instances and increase in relative importance of height is beneficial in another group of instances.

Unfortunately, it is not feasible to change the parameters every time another instance is solved, especially since it is not known if the parameter width or parameter height is preferred for the

currently solved instance. For this reason, another two tests introducing some randomisation for width and height were performed. The first randomized test selects the parameter for width at random from the interval $(0{,}75 - 1{,}25) \cup (2{,}75 - 3{,}25)$. The height used is then simply calculated as 4 – width i.e. the exact opposite of the width. The parameter for area is kept with the value of 1. The second randomized test is randomly switching between three different settings. First, area = 1, width = 1 and height = 1. Second, area = 1, width = 3 and height = 1 and third, area = 1, width = 1 and height = 3.

The results of the second randomized test reached relative distance from the optimum in best solutions of 4,12%, which were once again better than in any of the previously performed tests. After this it was decided that the strategy used in second randomized test will be used for this implementation. Since GRASP metaheuristic chooses only one best solution, the measure of the average over all the best solutions is more important than the average over all the solutions. The strategy "Random 2" outperforms all the other strategies in finding the best solutions, because it tries different strategies in different iterations and sometimes with big success. The fact that this strategy leads to worse average solutions is for GRASP not that important and it is a direct result of increased randomisation as sometimes less suitable criteria is being used.

5.4 Randomisation of the construction phase

The randomisation in the preceding preliminary tests was used in following way. First item in order was used with the probability of 70%, second with the probability of 20% and third with the probability of 10%. Now, another two increased randomisations will be tested. First, in which first item is used with the probability of 65%, second item with the probability of 20%, third item with the probability of 10% and fourth item with the probability of 5%. Second, in which first item is used with the probability of 60%, 25% probability is used for the second item, 10% for the third item and 5% for the fourth item. Both newly introduced randomisations represent an increase of randomisation.

		70%-20%-10%	65%-20%-10%-5%	60%-25%-10%-5%
Zero-Waste Instances	Avg. Solution	9,80%	10,08%	10,15%
	Avg. Best Solution	4,09%	4,07%	4,06%
Non-Zero-Waste Instances	Avg. Solution	9,24%	9,48%	9,59%
	Avg. Best Solution	4,19%	4,12%	4,09%
All Instances	Avg. Solution	9,63%	9,90%	9,99%
	Avg. Best Solution	4,12%	4,09%	4,07%

Table 4: Results primary testing randomisation of construction phase

Table 4, presented on previous page, summarizes the results of the relative distance from optimum for the three tested randomisation strategies. Just as expected the relative distance to the optimum of average of the best solutions increases to 4,09% for the second randomisation and 4,07% for the third randomisation reaching new best solutions, while the average over all the solution decreases with the increase in randomisation. Since the third strategy with the probabilities 60%-25%-10%-5% leads to the best solutions this strategy is chosen for the GRASP implementation. Further strategies with higher degree of randomisation are not tested, because the marginal increase of the best solutions is already very small. For this reason, it is assumed that further increase in randomisation would lead only to negligible improvement.

5.5 Heuristic for the Local Search Phase

Now, the preliminary tests for the local search phase will be concluded and this section deals with the choice of the repair heuristic. Following three repair heuristics were tested in the thesis. First, the Hybrid heuristic. Second, the bin packing version of the Hybrid heuristic and third, the bin packing version of the Adjusted Touching Perimeter heuristic. The results can be found in the Table 5 and it is important to mention that during the preliminary testing of the local search phase only a maximum of 200 solutions will be calculated for each instance. The reason for this is that during the local search naturally more solutions will be calculated as it represents an iterative process, where each of these 200 solutions consist of multiple iterations on its own. This change will increase the comparability with the results obtained previously during the construction phase.

		Without LS	HH	HH BinPack	ATP BinPack
Zero-Waste Instances	Avg. Solution	10,15%	8,25%	8,25%	7,35%
	Avg. Best Solution	4,06%	3,94%	4,04%	3,64%
Non-Zero-Waste Instances	Avg. Solution	9,59%	8,01%	8,04%	7,88%
	Avg. Best Solution	4,09%	3,95%	3,91%	4,02%
All Instances	Avg. Solution	9,99%	8,18%	8,19%	7,50%
	Avg. Best Solution	4,07%	3,94%	4,00%	3,75%

Table 5: Results primary testing repair heuristic

The Table 5, presents first the solutions obtained previously without the use of local search in the column labelled as “Without LS”, next the Hybrid Heuristic labelled as “HH”, then the bin packing version of the Hybrid Heuristic labelled as “HH BinPack” and last the bin packing version of the Adjusted Touching Perimeter labelled as “ATP BinPack”. The first thing clear from the Table 5 is that the implementing of LS improves the overall performance. There is an improvement of at least 0,07% in

average over all the best solutions and of at least 1,8% of all the solutions. This means that the local search not only repaired some of the previously created poorer solutions, but even managed to lead to new best solutions. Next, the bin packing version of the Adjusted Touching Parameter performs in general better than the rest of the repair heuristics closing the gap to the optimum to 3,75% over the all the best solutions found. For this reason, the bin packing version of the Adjusted Touching Perimeter is subsequently used for this implementation of GRASP metaheuristic.

5.6 Randomisation of repair heuristic for the Local Search Phase

The items packed back inside of the container during the repair should be taken in a slightly changed order, so that every iteration of local search explores another part of the local neighbourhood. During the preliminary testing two different procedures were considered. First, choosing the first three items in a randomised manner, where the randomisation is identical with the randomisation used during the construction phase. The rest of the items is packed in original order. Second, similar to the first process in the randomisation of the items, but additionally banning the first two items from being packed in the beginning. This means that for the first three packed items there is a chance 60% that the third item will be packed, 25% that the fourth item will be packed, 10% that the fifth item will be packed and 5% that the sixth item will be packed. The rest of the items are again packed in original order. Both randomisations are visualized in the Table 6, where the exact probability with which each of the items could be packed back are listed. It is important to notice once again that this probability distribution will be only used during the packing of the first three items after what the previous packing order will be resumed. This means that if the first three packed items were the items number 2, 4 and 3, the next packed item will be the item number 1, then the item number 5, 6 and so on.

Original Packing Order	1	2	3	4	5	6	7
Randomisation	60%	25%	10%	5%	0%	0%	0%
Banning + Randomisation	0%	0%	60%	25%	10%	5%	0%

Table 6: Change in order in which the items are packed back inside of the container

The results of this part of primary testing are presented in the Table 7 on the next page. The banning of the first two packed items was implemented in order to avoid the creation of the same holes. The results obtained from this procedure are on average 0,05% worse than the randomisation only procedure. Overall it seems that both tested procedures have similar performance. It seems that it is not necessary to try to avoid the creation of the same holes, because different heuristic is used for construction and local search which effects the placement of items inside of the container and

automatically leads to change in the pattern. For this reason and because the "randomisation only" strategy performed on average slightly better on average than "Banning + Randomisation" strategy the "randomisation only" strategy was chosen for the implementation.

		Banning + Randomization	Randomization
Zero-Waste Instances	Avg. Solution	7,25%	7,12%
	Avg. Best Solution	3,62%	3,53%
Non-Zero-Waste Instances	Avg. Solution	7,80%	7,95%
	Avg. Best Solution	3,91%	3,93%
All Instances	Avg. Solution	7,41%	7,36%
	Avg. Best Solution	3,70%	3,65%

Table 7: Results primary testing randomisation of the local search phase

5.7 The partial destruction in the Local Search Phase

The preliminary tests performed in this section will explore how the decision of which hole should be chosen for the unpacking influences the performance. As already previously mentioned after the holes in the pattern were identified they are sorted according to their area and one of the biggest holes will be chosen for unpacking at random. How many of the biggest holes are included in the random selection influences the variance of the partial destruction and this affects the performance of the GRASP metaheuristic. The Table 8 represents three different strategies tested in this section and each of this strategies are named according to how many of the holes were considered at random for the selection. This means that three strategies are considered, first taking only one of the two biggest holes in consideration second, taking three biggest holes in consideration or third, taking four of the biggest holes in consideration. Naturally, when more holes are taken in consideration the variance of the Large Neighbourhood Search increases.

		2 Biggest	3 Biggest	4 Biggest
Zero-Waste Instances	Avg. Solution	7,35%	7,25%	7,29%
	Avg. Best Solution	3,64%	3,62%	3,63%
Non-Zero-Waste Instances	Avg. Solution	7,88%	7,80%	7,88%
	Avg. Best Solution	4,02%	3,91%	4,02%
All Instances	Avg. Solution	7,50%	7,41%	7,46%
	Avg. Best Solution	3,75%	3,70%	3,74%

Table 8: Results primary testing partial destruction

The Table 8, presented on the previous page, once again presents the relative distances from the optimum. The option of choosing one of the three biggest holes leads to an overall improvement of about 0,05% in comparison to the other two options. This means that choosing only between two biggest holes restricts the local search from finding some of the better solutions and choosing four or more of the biggest holes makes the local search too broad in order to find the better solutions effectively. After this preliminary test it was clear that choosing one of the three biggest holes is optimal for this implementation of the GRASP metaheuristic.

5.8 The intensity of local search

In this section of the preliminary testing it will be investigated how many times should the local search be repeated, before it is decided that the currently searched neighbourhood of solutions will probably not lead to further improvements and a new iteration of GRASP metaheuristic should be started. The depth of local search highly influences the quality and the computation time of implemented GRASP metaheuristic. As Feo & Resende (1995) describe in their work, the iterative sampling of the GRASP metaheuristic represents a strategy in order to avoid the entrapment at local optima. This means that the local search should extend over enough iterations to find the local optimum, but once the local optimum is found another iteration of GRASP metaheuristic should be started in order to find the global optimum through this iterative sampling. This situation although creates a problem of identifying the current solution as the local optimum. The solution of this problem is terminating the local search when no better solutions are found multiple times in succession.

When the local search doesn't find a better solution two possible reasons for this can be considered, either the current solution represents already the local optimum, which cannot be further improved with local search, or the local search resulted in a suboptimal solution and the local search should be repeated. It is impossible to know if repeating the local search can yield to further improvements and at this point three strategies are being tested, with regards to when it is best to declare, that the local search is stuck in the local optimum and a new GRASP iteration should be started.

First strategy is waiting until the local search cannot find a better solution two times in succession and then terminating the local search. Second strategy is waiting until the local search doesn't find any better solution three times in succession and third strategy waits until there was no better solution for four times in succession, before terminating the local search. Naturally, every next strategy gives the algorithm bigger chance to find the local optimum but takes longer time to compute. It is the goal of this testing to compare the marginal improvement of waiting longer to the total

computation efficiency. The preliminary testing of these three strategies was concluded only on some of the instances used during the previous preliminary testing. The reason behind this was to maximize the comparability when comparing these three strategies, because the depth of local search directly influences the implied number of solutions. For this reason, only the bigger instances where the time limit (60 seconds) and not the number of solutions was the stopping criterion were taken.

The Table 9 represents the results over all the considered instances and it can be seen that waiting until there are three successive cases of no better solution resulted in improvement of the best reached solutions of 0.172% in comparison to terminating the local search after two successive cases of no better solution. On the other hand, the marginal improvement of waiting until there are four successive cases of no better solution were only 0.002%. This means that the highest dept of local search explored generated best results, but these results were only marginally better than with three successive cases. Because of this a T-Test of the results converted from a log-normal to a normal distribution was used which resulted in value of 11.8% showing that there is no significant difference between the results obtained with three and four successive cases of no better solution. This implies that the deepest local search is unnecessary, because it is already in most of the cases stuck in the local optimum and the computation time can be used more effectively if only three successive cases of no better solution will be considered. Consequently, the GRASP heuristic will be implemented so that three successive cases of no better solution will lead to termination of the local search and start of next GRASP iteration.

	Two Successive Cases	Three Successive Cases	Four Successive Cases
Avg. Solution	3,985%	3,751%	3,572%
Avg. Best Solution	2,554%	2,382%	2,380%
T-Test	0,11%		11,80%

Table 9: Results primary testing depth of local search

6 Testing, results and conclusion

This is the last chapter of this thesis where all the missing parts will be concluded. The finalized GRASP metaheuristic was tested on all the instances presented in the beginning of the previous chapter and each of the instances was solved for a total of 10 times. The implementation was executed using the Microsoft Visual Studio 2017 on a computer equipped with processor Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz and 8 GB of RAM memory. The stopping criteria used for the implemented GRASP metaheuristic were either competition of 50 000 iterations or reaching a time limit of 60 seconds. This stopping criteria was chosen to be the same as in other papers like the work of Zhang, Che, Ye, Si, & Leung (2016) or Hopper & Turton (2001) to enable the comparison of results.

This chapter consists of following sections. First section, presenting other metaheuristics from the literature, which the implemented GRAPS metaheuristic is compared to. Next section, presenting the results of the final testing in tables containing results of this tests plus the results from other papers and final section, containing the conclusion and drafting possible research extensions to the future.

6.1 Compared Metaheuristics and Heuristics

The reactive GRASP previously implemented by Alvarez-Valdes, Parreño, & Tamarit (2008) is according to Zhang, Che, Ye, Si, & Leung (2016) one of heuristics that are even presently considered excellent. For this reason, the best metaheuristic approaches were chosen in order to benchmark the implemented GRASP metaheuristic properly. Additionally, two approaches are included into the comparison as well. First, a fairly new approach by Babaoğlu (2017) showcasing that not all new algorithms in literature exhibit excellent results and second, an iterative randomised implementation of the Adjusted Touching Perimeter presented in this thesis to benchmark the implemented GRASP metaheuristic against a simplified version of itself.

This paragraph presents the compared metaheuristics from other papers listed in chronological order. This ordering is kept for all the tables containing the final results positioning newer papers more to the left side of the table. First presented metaheuristic is the fruit fly optimilization presented by Babaoğlu (2017). This heuristic will be presented only in the Table 10 under the name Fruit Fly. Second heuristic is the hybrid algorithm from Zhang, Che, Ye, Si, & Leung (2016), which represents a combination of variable neighbourhood search and an improved heuristic

algorithm. This algorithm is in results section dubbed as Hybrid VNS. Third heuristic is a binary search heuristic using randomised local search presented by Zhang, Wei, Leung, & Chen (2013). Here only the part of results, where the items cannot be rotated, is compared. Fourth heuristic is a simple randomized algorithm (SRA) presented in the work of Yang, Han, & Ye (2013). Fifth heuristic is an intelligent search algorithm (ISA) enhancing the metaheuristic simulated annealing with a multi-start strategy presented by Leung, Zhang, & Mong Sim (2011). Sixth heuristic is a heuristic for one dimensional packing adopted for two dimensional packing dubbed as SVC presented by Belov, Scheithauer, & Mukhacheva (2008) and finally the last heuristic is the reactive GRASP by Alvarez-Valdes, Parreño, & Tamarit (2008) which was presented already during the introduction.

The iterative randomised Adjusted Touching Perimeter, which represents a simplification of the GRASP metaheuristic implement in this thesis, is implemented in following way. First, the Hybrid heuristic from this thesis is used to create an upper bound. Then, the bin packing version of the Adjusted Touching Perimeter is used in iterative fashion, while the maximal height of the strip is always one unit shorter than the previously established upper bound. If the Adjusted Touching Perimeter finds a feasible solution, this solution is saved and the upper bound is updated. Then, as many iterations of Adjusted Touching Perimeter are completed as possible. This means that during this implementation no local search is performed. The time limit, randomisation and other settings are identical to the implemented GRASP metaheuristic. This simple implementation shows that the randomized Adjusted Touching Perimeter performs very well, given a good upper bound. This heuristic will be dubbed in the results as TP and can be found left of the results of the originally implemented GRASP metaheuristic.

6.2 Results Zero-Waste Instances

Now the results from the zero-waste instances of the testing will be summarised and presented with the help of two measurements. First, the average of relative distances from the optimum and second the relative distance of the best reached solution from the optimum. These results can be found in Table 10 - Table 15 located on the following pages and contain the results of the compared metaheuristics and heuristics presented in the previous section. In addition to these results visualizations of some sample packings can be found in Appendix. These visualisations mostly present the worst performing packings uncovering the reasons for the underperformance of the implemented GRASP metaheuristic and the reader will be pointed to these at appropriate time in order to support some of the interpretations of results presented in this chapter.

Instance				TP		Our GRASP		Fruit Fly	Binary Search		SRA		ISA		SVC	Reactive GRASP	
	n	*W*	*LB*	*avg.*	*best*	*avg.*	*best*	*avg.*	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*	*avg.*	*avg.*	*best*
C1	16-17	20	20	1,7%	1,7%	0,7%	**0,0%**	10,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	1,7%	0,0%	0,0%
C2	25	40	15	1,3%	**0,0%**	2,9%	**0,0%**	46,7%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
C3	28-29	60	30	3,0%	2,2%	3,0%	2,2%	20,0%	1,1%	1,1%	1,1%	1,1%	1,1%	1,1%	1,1%	1,1%	1,1%
C4	49	60	60	**1,7%**	**1,7%**	2,2%	**1,7%**	11,7%	1,6%	1,1%	1,7%	1,7%	1,6%	1,1%	1,7%	1,7%	1,7%
C5	73	60	90	1,4%	**1,1%**	1,7%	**1,1%**	7,8%	1,0%	0,7%	0,8%	0,4%	1,0%	0,6%	1,1%	1,1%	1,1%
C6	97	80	120	**1,4%**	1,4%	1,5%	1,1%	5,8%	0,9%	0,8%	0,8%	0,8%	0,8%	0,8%	1,4%	1,6%	0,8%
C7	196-197	160	240	1,5%	**1,3%**	1,7%	1,5%	2,5%	0,5%	0,4%	0,4%	0,4%	0,7%	0,7%	1,1%	1,4%	1,3%
Average Gap				1,7%	1,3%	1,9%	1,1%	14,9%	0,7%	0,6%	0,7%	0,6%	0,7%	0,6%	1,2%	1,0%	0,9%

Table 10: Results from instances of Hopper & Turton (2001)

Table 10 contains the results of 21 instances from Hopper & Turton (2001). These instances are, together with the 70 instances from Hopper E. (2000) in the Table 11, which can be found on the next page, most commonly used zero-waste-instances in the literature. The average of the best results of the instances from Hopper & Turton (2001) produced solutions with relative distance of 1.1% above the optimal solution, which shows that the implemented GRASP metaheuristic has an overall performance only slightly worse than the Reactive GRASP from Alvarez-Valdes, Parreño, & Tamarit (2008) and in fact reaches the same quality of solution in more than half of these 21 tested instances. These special cases where the implemented heuristic lead to equal performance are highlighted with colour. The rest of the compared heuristics, except the Fruit Fly Optimization, outperform the implemented GRASP metaheuristic and result in about 50% less waste. The results of the Fruit Fly Optimization are very poor and can be compared only to results of some very primitive algorithms. The implemented GRASP metaheuristic further shows higher spread between the best reached results and the average of all the results. This spread, which is for most of the compared heuristics only about 0.1%, reaches 0.7% for the implemented GRASP metaheuristic pointing to possible inconsistency in reaching the best achievable solution.

The results from the Table 11, presented on the next page, reveal a similar story to the previously presented Table 11 and the difference of the best solutions between the implemented GRASP metaheuristic and the Reactive GRASP from Alvarez-Valdes, Parreño, & Tamarit (2008) is on average only 0.4% points from the optimum, but the spread between the best reached solution and the average of all the solutions is on average 0.6% points higher. Only in the case of the big non-guillotineable instances N6 and N7 from Hopper E. (2000) the implemented GRASP metaheuristic outperformed the Reactive GRASP from Alvarez-Valdes et al. (2008) on average by 0.15%.

Interestingly, the results from Table 10 and Table 11 show that the randomized Adjusted Touching Perimeter performs in 57% of the cases better than the implemented GRASP metaheuristic, even though it represents a simpler algorithm. There are many cases in these instances by Hopper &

Turton (2001) and Hopper E. (2000), when the randomized Adjusted Touching Perimeter even performs comparably and in some cases actually better than the Reactive GRASP from Alvarez-Valdes,Parreño, & Tamarit (2008) and these can be found highlighted in colour. The Adjusted Touching Perimeter even reaches a 0.1% better average best solution for the section of guillotineable instances from Hopper E. (2000) than the Reactive GRASP from Alvarez-Valdes et al. (2008).

The rest of the compared heuristics once again outperform both the implemented GRASP metaheuristic and randomized Adjusted Touching Perimeter heuristic, but it is once again important to mention that Zhang, Che, Ye, Si, & Leung (2016) in their quite recent paper still regard the results from Alvarez-Valdes et al. (2008) as excellent and this means that the achieved results can be so far described as much better than satisfactory.

Instance				TP		Our GRASP		Hyb. VNS	SRA		ISA		SVC	Reactive GRASP	
	n	*W*	*LB*	*avg.*	*best*	*avg.*	*best*	*avg.*	*avg.*	*best*	*avg.*	*best*	*avg.*	*avg.*	*best*
T1	17	200	200	2,0%	0,0%	0,8%	0,0%	0,0%	0,0%	0,0%	1,2%	1,0%	0,9%	0,0%	0,0%
T2	25	200	200	3,9%	3,3%	4,6%	3,9%	2,2%	2,6%	1,7%	3,7%	2,8%	3,5%	3,2%	3,2%
T3	29	200	200	4,1%	**3,5%**	4,6%	4,1%	2,3%	2,2%	1,6%	3,5%	3,4%	3,3%	3,7%	3,7%
T4	49	200	200	**3,0%**	**2,6%**	3,7%	3,3%	2,3%	2,6%	2,0%	2,7%	2,4%	2,5%	3,0%	2,7%
T5	73	200	200	2,6%	2,3%	3,3%	2,9%	1,8%	2,3%	1,9%	2,2%	2,1%	2,1%	2,4%	2,2%
T6	97	200	200	**1,9%**	**1,7%**	2,3%	2,1%	1,4%	1,7%	1,5%	1,7%	1,3%	1,6%	2,1%	1,9%
T7	199	200	200	1,8%	1,6%	2,2%	1,9%	0,5%	0,5%	0,5%	0,7%	0,6%	1,0%	1,5%	1,4%
Average Gap				2,7%	2,1%	3,1%	2,6%	1,5%	1,7%	1,3%	2,2%	1,9%	2,1%	2,3%	2,2%
N1	17	200	200	4,0%	3,8%	3,9%	1,8%	0,0%	0,0%	0,0%	1,1%	1,0%	3,3%	0,9%	0,9%
N2	25	200	200	4,2%	3,7%	4,8%	4,0%	0,9%	1,2%	0,5%	3,2%	3,2%	3,4%	3,3%	3,3%
N3	29	200	200	4,3%	3,7%	4,7%	4,3%	2,0%	2,0%	1,8%	3,4%	3,2%	3,5%	3,6%	3,6%
N4	49	200	200	**2,9%**	**2,6%**	3,5%	3,0%	2,2%	2,7%	2,3%	2,8%	2,6%	2,5%	3,0%	2,8%
N5	73	200	200	2,8%	2,4%	3,2%	2,9%	1,8%	2,2%	1,9%	2,3%	1,9%	2,1%	2,6%	2,3%
N6	97	200	200	**2,2%**	**1,9%**	2,4%	**2,0%**	1,2%	1,5%	1,3%	1,7%	1,4%	1,7%	2,2%	2,1%
N7	199	200	200	**1,1%**	**1,0%**	**1,2%**	**1,0%**	0,5%	0,5%	0,5%	0,9%	0,6%	1,0%	1,3%	1,2%
Average Gap				3,1%	2,7%	3,4%	2,7%	1,2%	1,4%	1,2%	2,2%	2,0%	2,5%	2,4%	2,3%

Table 11: Results from instances of Hopper (2000)

The next set of results, in the Table 12 on the next page, represents the results of a smaller set of only 13 instances from Burke, Kendall, & Whitwell (2004). This set of instances is interesting, because it incorporates even instances comprised of as many as 500 and 3000 items. The overall results are remarkable with the relative distance from the optimum on average only 0.1% higher than the results of Alvarez-Valdes, Parreño, & Tamarit (2008). The comparison to the rest of the metaheuristics reveals that these instances overall reach solutions with smaller relative distance to the optimum, where the simple randomized algorithm presented by Yang, Han, & Ye (2013) reached the relative distance as small as 0.1% from the optimum.

Instance				TP		Our GRASP		Binary Search		SRA		ISA		SVC	Reactive GRASP	
	n	*W*	*LB*	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*	*avg.*	*avg.*	*best*
N1	10	40	40	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
N2	20	30	50	2,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
N3	30	30	50	2,0%	2,0%	2,0%	2,0%	2,0%	2,0%	0,0%	0,0%	0,2%	0,0%	0,0%	2,0%	2,0%
N4	40	80	80	1,3%	1,3%	1,3%	1,3%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	1,3%	1,3%	1,3%
N5	50	100	100	3,0%	3,0%	3,0%	3,0%	0,0%	0,0%	0,0%	0,0%	1,0%	1,0%	1,0%	2,0%	2,0%
N6	60	50	100	1,3%	1,0%	1,0%	1,0%	0,0%	0,0%	0,2%	0,0%	0,9%	0,0%	1,0%	1,0%	1,0%
N7	70	80	100	1,0%	1,0%	1,0%	1,0%	0,0%	1,0%	0,0%	0,0%	0,0%	0,0%	1,0%	1,0%	1,0%
N8	80	100	80	1,6%	1,3%	2,3%	1,3%	1,3%	0,0%	1,3%	1,3%	1,3%	1,3%	1,3%	1,3%	1,3%
N9	100	50	150	1,3%	1,3%	1,3%	1,3%	0,0%	0,0%	0,6%	0,0%	0,6%	0,0%	0,7%	0,7%	0,7%
N10	200	70	150	0,8%	0,7%	0,8%	0,7%	0,9%	0,7%	0,4%	0,0%	0,5%	0,0%	0,7%	0,7%	0,7%
N11	300	70	150	1,3%	0,7%	1,3%	0,7%	0,0%	0,0%	0,1%	0,0%	0,5%	0,0%	0,7%	0,7%	0,7%
N12	500	100	300	2,4%	2,0%	1,9%	1,7%	0,3%	0,3%	0,3%	0,3%	0,3%	0,3%	0,3%	1,3%	1,3%
N13	3152	640	960	0,6%	0,6%	0,4%	0,4%	0,1%	0,0%	0,1%	0,1%	0,0%	0,0%	0,3%	0,5%	0,5%
Average Gap				1,3%	1,1%	1,2%	1,1%	0,4%	0,3%	0,2%	0,1%	0,4%	0,2%	0,6%	1,0%	1,0%

Table 12: Results from instances of Burke et al. (2004)

The two highlighted instances N12 and N13 are interesting, as only one iteration of the implemented GRASP metaheuristic could be calculated in these cases. The reason for this was that the calculation of this one iteration took 120 – 220 seconds for the instance N12 containing 500 rectangles and astonishing 4200 seconds for the instance N13 containing 3152 rectangles. This results reveal a weakness of the Adjusted Touching perimeter heuristic and Hybrid Heuristic used in the implemented GRASP metaheuristic in comparison to the much quicker best fit heuristic employed by Alvarez-Valdes et al. (2008). It is even more interesting that despite of this the implemented GRASP metaheuristics achieved result in the instance N13 0.1% better than the result of Reactive GRASP implemented by Alvarez-Valdes, Parreño, & Tamarit (2008).

Instance				TP		Our GRASP		Binary Search		SRA		ISA		SVC	Reactive GRASP	
	n	*W*	*LB*	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*	*avg.*	*avg.*	*best*
Nice1	25	100	100	5,1%	4,8%	5,6%	4,7%	4,6%	4,4%	3,7%	3,5%	4,1%	3,5%	3,7%	3,4%	3,4%
Nice2	50	100	100	8,4%	7,9%	9,6%	8,1%	3,0%	2,8%	3,7%	3,0%	4,6%	4,2%	3,7%	4,6%	4,6%
Nice3	100	100	100	8,6%	7,8%	8,9%	7,9%	3,0%	2,8%	3,3%	3,1%	3,5%	2,8%	3,4%	4,0%	4,0%
Nice4	200	100	100	11,0%	9,9%	9,9%	9,2%	1,9%	1,6%	2,2%	2,2%	2,9%	2,9%	2,5%	3,6%	3,6%
Nice5	500	100	100	9,0%	9,0%	6,7%	6,7%	1,0%	0,9%	0,8%	0,8%	1,5%	1,5%	1,7%	2,4%	2,4%
Nice6	1000	100	100	8,9%	8,9%	7,5%	7,5%	0,5%	0,4%	0,3%	0,3%	1,2%	1,2%	1,5%	2,1%	2,1%
Path1	25	100	100	4,7%	4,2%	5,2%	4,2%	4,2%	4,2%	4,2%	4,2%	4,1%	4,1%	4,1%	4,1%	4,1%
Path2	50	100	100	2,8%	2,6%	3,2%	2,8%	1,0%	0,8%	0,9%	0,1%	1,5%	1,2%	1,4%	1,9%	1,9%
Path3	100	100	100	3,0%	2,9%	3,5%	3,1%	4,3%	4,0%	2,3%	2,2%	2,3%	2,0%	2,2%	2,7%	2,7%
Path4	200	100	100	4,3%	3,8%	3,8%	3,4%	5,2%	5,1%	1,4%	1,4%	1,6%	1,5%	1,6%	2,1%	2,1%
Path5	500	100	100	5,5%	5,2%	4,5%	4,5%	3,0%	2,9%	1,4%	1,4%	2,0%	2,0%	2,2%	3,4%	3,4%
Path6	1000	100	100	5,9%	5,9%	5,0%	5,0%	2,1%	1,5%	0,6%	0,6%	0,9%	0,9%	2,1%	2,4%	2,4%
Average Gap				6,4%	6,1%	6,1%	5,6%	2,8%	2,6%	2,1%	1,9%	2,5%	2,3%	2,5%	3,1%	3,1%

Table 13: Results from instances Wang & Valenzela (2001)

The Table 13, presented on previous page, is containing the 12 instances from Wang & Valenzela (2001) reveals similar story to the Table 12, where only one iteration of GRASP could be solved for the highlighted bigger instances comprised of 500 and 1000 items. Other than this the implemented metaheuristics perform worse than the previously presented instances producing in many cases more than twice the waste than the compared metaheuristics from the literature. This loss in performance can be accounted to the loss of variation in bigger instances, where only a limited number of GRASP iterations could be completed, but this doesn't explain the underperformance of some of the smaller instances like Nice2, Nice3 and Nice4. Visually comparing the results of packing the instances Nice4 and Nice3, presented in Figure 34 in Appendix on the page 48, to other sample packings presented in Appendix reveals that the implemented metaheuristics don't deal well with problems where the variation in size of items is small. It seems that for items of similar size the use of different packing heuristic is preferable.

6.3 Results Non-Zero-Waste Instances

The next set of instances presented in the Table 14 is from the authors Beasley (1985a) and (1985b),Christofides & Whitlock (1977) and Bengtsson (1982). These instances already represent non-zero-waste problems and the lower bound used to compute the relative distances for this Table 14 as well as later for the Table 15 were adapted from the work of Iori, Martello, & Monaci (2003). This way of calculating the relative distances was used in compared papers as well.

Instance		TP		Our GRASP		SRA		ISA		SVC	Reactive GRASP	
	Number of instances	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*	*avg.*	*avg.*	*best*
cgcut	3	1,8%	1,8%	3,0%	2,8%	3,9%	3,8%	2,3%	2,2%	2,4%	2,4%	2,4%
gcut	13	8,7%	8,2%	8,1%	7,8%	5,9%	5,9%	6,0%	5,9%	6,3%	6,1%	6,1%
ngcut	12	3,0%	2,8%	1,5%	1,5%	2,1%	2,1%	2,2%	2,2%	1,8%	1,3%	1,3%
beng	10	1,0%	0,5%	1,1%	0,7%	0,3%	0,3%	0,3%	0,3%	0,0%	0,0%	0,0%
Average Gap		3,6%	3,3%	3,4%	3,2%	3,0%	3,0%	2,7%	2,7%	2,6%	2,4%	2,4%

Table 14: Results from instances of Beasley (1985a), Beasley (1985b), Christofides & Whitlock (1977) and Bengtsson (1982)

It was assumed that the employed construction heuristics using the touching perimeter as criteria for placement of the items, will perform worse for the non-zero-waste problems than the zero-waste problems. The assumption was that the holes naturally contained inside of the pattern will lead to errors, but results presented in the Table 14 show that this is not the case and the implemented heuristics perform slightly better than in the previously presented zero-waste problems. This can be seen especially on the three instances from Christofides & Whitlock (1977) named cgcut, where the

randomized Adjusted Touching Perimeter outperforms by a margin of at least 0.4% all other metaheuristics from the literature and the 12 instances from Beasley (1985a) where the implemeted GRASP metaheuristic outperforms the simple randomized algorithm (SRA) from Yang, Han, & Ye (2013) the intelligent search algorithm (ISA) from Leung, Zhang, & Mong Sim (2011) and the SVC heuristic from Belov, Scheithauer, & Mukhacheva (2008) all of which exhibited exceptionally good results up until this point. These results combined with good performance of Reactive GRASP from Alvarez-Valdes et al. (2008) suggest that GRASP is suitable for solving the non-zero-waste packing problems.

The results of the non-zero-waste instances from authors Berkey & Wang (1987) and Martello & Vigo (1998) presented in the Table 15 are much worse in comparison to the previously presented non-zero-waste instances presented in the Table 14. The assumed reason is that the containers of these tested instances were often very long and slim, which rendered the gradual adjusting of the upper bound ineffective. When inspecting the sample packings presented in Figure 36 on the page 49 it is obvious that in these instances the initial packing order is very important as often only one item can be fitted in one row which makes the destroy and repair mechanism of the local search ineffective.

The randomized Adjusted Touching Perimeter once again leads in some cases to better performance than the implemented GRASP metaheuristic. These cases are in the Table 15 highlighted with darker colour. The reason for this could simply be that the gradual adjusting of the upper bound is given more time in randomized Adjusted Touching Perimeter and by skipping the local search the algorithm has better chances in randomly finding a better packing order.

Instance		TP		Our GRASP		Hyb. VNS	SRA		ISA		SVC	Reactive GRASP	
	W	*avg.*	*best*	*avg.*	*best*	*avg.*	*avg.*	*best*	*avg.*	*best*	*avg.*	*avg.*	*best*
C01	10	4,6%	4,4%	4,6%	4,5%	0,6%	0,6%	0,6%	0,6%	0,6%	0,6%	0,6%	0,6%
C02	30	0,6%	0,5%	0,8%	0,5%	0,2%	0,2%	0,1%	0,2%	0,1%	0,1%	0,2%	0,2%
C03	40	7,7%	7,3%	7,6%	7,1%	1,6%	1,7%	1,5%	1,7%	1,6%	1,9%	1,8%	1,8%
C04	100	3,0%	2,6%	3,5%	3,0%	1,5%	1,6%	1,4%	1,8%	1,5%	1,9%	2,1%	2,1%
C05	100	10,9%	10,2%	10,1%	9,7%	2,0%	2,1%	2,0%	2,0%	2,0%	2,3%	2,1%	2,1%
C06	10	**4,3%**	**3,9%**	5,0%	4,4%	2,8%	3,1%	2,7%	3,3%	2,9%	3,1%	3,2%	3,2%
C07	30	16,1%	15,8%	15,8%	15,6%	1,1%	1,1%	1,1%	1,1%	1,1%	1,1%	1,1%	1,1%
C08	40	5,6%	5,0%	5,2%	4,6%	3,1%	3,3%	3,0%	3,2%	2,9%	3,7%	3,7%	3,7%
C09	100	24,4%	24,4%	24,4%	24,4%	0,1%	0,1%	0,1%	0,1%	0,1%	0,1%	0,1%	0,1%
C10	100	5,4%	5,0%	5,4%	5,0%	2,6%	2,6%	2,5%	2,7%	2,6%	3,2%	3,0%	3,0%
Average Gap		8,3%	7,9%	8,2%	7,9%	1,6%	1,6%	1,5%	1,7%	1,5%	1,8%	1,8%	1,8%

Table 15: Results from instances of Berkey & Wang (1987) and Martello & Vigo (1998)

6.4 Conclusion

The results presented in the previous section lead to these conclusions. First, the GRASP metaheuristic presented and implemented in this thesis represents a viable option for solving various strip packing problems. The strong points of this approach are simplicity and reasonably good performance. The results have further shown that further simplified version of the implemented GRASP metaheuristic managed to deliver surprisingly good results. Next, the implementation of simplified version has further shown that the randomisation combined with iterative approach was the strongest aspect influencing the quality of results of the implemented GRASP metaheuristic. Last, three weak points were identified. First, the sluggishness of the implemented heuristics, resulting in poor results for instances containing more than 300 items. Second, the underperformance in instances containing items of similar size and third, the infectivity in search for upper bounds causing infectivity of the local search for instances consisting of long and slim strips. This infectivity was vital, because of the use of bin packing version of the Adjusted Touching Perimeter heuristic in local search, which performance depends on the choice of the upper bound greatly.

The future research on this topic should include following three parts. First, implementing improved version of the iterative randomized Adjusted Touching Perimeter heuristic. Proper preliminary testing and suit tailored parameters could make this heuristic outperform the implemented GRASP metaheuristic. Second, a method to speed up the implemented GRASP metaheuristic should be explored. Improving the effectivity of implementation or parallel processor computing could prove helpful in solving this problem by allowing for multiple iterations to be performed simultaneously. Third, a more effective method for determining upper bounds could be implemented. Currently the upper bound is in every iteration adjusted by only one point, which seems to be especially ineffective for the higher instances. The search for an optimal upper bound could incorporate computing a theoretical lower bound of the solution, subsequently followed by exploration of the points between the lower and the upper bound. This search could incorporate some version of Fibonacci search algorithm, which example could be found in the work by Ramaprabha, Balaji, & Mathur (2012). This would improve the operation of the bin packing version of the Adjusted Touching Perimeter heuristic and subsequently the performance for very high strips.

7 References

Alvarez-Valdes, R., Parreño, F., & Tamarit, J. (2008). Reactive GRASP for the strip-packing problem. *Computers & Operations Research, 35*(4), 1065-1083.

Babaoğlu, İ. (2017). Solving 2D strip packing problem using fruit fly optimization algorithm. *Procedia Computer Science, 111*, 52-57.

Bang-Jensen, J., Gutin, G., & Yeo, A. (2004). When the greedy algorithm fails. *Discrete Optimization, 1*(2), 121-127.

Beasley, J. (1985a). Algorithms for Unconstrained Two-Dimensional Guillotine Cutting. *Journal of the Operational Research Society, 36*(4), 297-306.

Beasley, J. (1985b). An Exact Two-Dimensional Non-Guillotine Cutting Tree Search Procedure. *Operations Research, 33*(1), 49-64.

Belov, G., Scheithauer, G., & Mukhacheva, E. (2008). One-dimensional heuristics adapted for two-dimensional rectangular strip packing. *Journal of the Operational Research Society, 59*(6), 823-832.

Bengtsson, B.-E. (1982). Packing Rectangular Pieces--A Heuristic Approach. *The Computer Journal, 25*(3), 353-357.

Berkey, J., & Wang, P. (1987). Two-Dimensional Finite Bin-Packing Algorithms. *The Journal of the Operational Research Society, 38*(5), 423.

Burke, E., & Kendall, G. (1999). Applying Simulated Annealing and the No Fit Polygon to the Nesting Problem. *PROCEEDINGS OF THE WORLD MANUFACTURING CONGRESS*, (pp. 70–76). Durham, UK.

Burke, E., Kendall, G., & Whitwell, G. (2004). A New Placement Heuristic for the Orthogonal Stock-Cutting Problem. *Operations Research, 52*(4), 655-671.

Chazelle, B. (1983). The Bottomn-Left Bin-Packing Heuristic: An Efficient Implementation. *IEEE Transactions on Computers, C-32*(8), 697-707.

Christofides, N., & Whitlock, C. (1977). An Algorithm for Two-Dimensional Cutting Problems. *Operations Research, 25*(1), 30-44.

Feo, T., & Resende, M. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization, 6*(2), 109-133.

Feo, T., Resende, M., & Smith, S. (1994). A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. *Operations Research, 42*(5), 860-878.

Hopper, E. (2000). *Two-dimensional Packing utilising Evolutionary Algorithms and other Meta-Heuristic Methods.* Ph.D. thesis, University of Wales, Cardiff.

Hopper, E., & Turton, B. (1999). A genetic algorithm for a 2D industrial packing problem. *Computers & Industrial Engineering, 37*(1-2), 375-378.

Hopper, E., & Turton, B. (2001). An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research, 128*(1), 34-57.

Iori, M., Martello, S., & Monaci, M. (2003). Metaheuristic Algorithms for the Strip Packing Problem. In V. Korotkikh, & P. M. Pardalos (Eds.), *Optimization and Industry: New Frontiers* (pp. 159-179). Springer, Boston, MA.

Jakobs, S. (1996). On genetic algorithms for the packing of polygons. *European Journal of Operational Research, 88*(1), 165-181.

Jylänki, J. (2010). *A Thousand Ways to Pack the Bin-A Practical Approach to Two-Dimensional Rectangle Bin Packing*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.695.2918&rep=rep1&type=pdf

Leung, S., Zhang, D., & Mong Sim, K. (2011). A two-stage intelligent search algorithm for the two-dimensional strip packing problem. *European Journal of Operational Research, 215*(1), 57-69.

Lodi, A., Martello, S., & Monaci, M. (2002). Two-dimensional packing problems: A survey. *European Journal of Operational Research, 141*(2), 241 - 252.

Lodi, A., Martello, S., & Vigo, D. (1999). Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems. *INFORMS Journal on Computing, 11*(4), 345-357.

Martello, S., & Vigo, D. (1998). Exact Solution of the Two-Dimensional Finite Bin Packing Problem. *Management Science, 44*(3), 388-399.

Pisinger, D., & Ropke, S. (2010). Large Neighborhood Search. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 399-419). Boston, MA: Springer US.

Prais, M., & Ribeiro, C. (2000). Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment. *INFORMS Journal on Computing, 12*(3), 164-176.

Ramaprabha, R., Balaji, M., & Mathur, B. (2012). Maximum power point tracking of partially shaded solar PV system using modified Fibonacci search method with fuzzy controller. *International Journal of Electrical Power & Energy Systems, 43*(1), 754-765.

Resende, M., & Ribeiro, C. (2003). Greedy Randomized Adaptive Search Procedures. In F. Glover, & G. Kochenberger (Eds.), *Handbook of Metaheuristics* (pp. 219-249). Boston: Kluwer Academic Publishers.

Wang, P., & Valenzela, C. (2001). Data set generation for rectangular placement problems. *European Journal of Operational Research, 134*(2), 378-391.

Yang, S., Han, S., & Ye, W. (2013). A simple randomized algorithm for two-dimensional strip packing. *Computers and Operation Research, 40*(1), 1-8.

Zhang, D., Che, Y., Ye, F., Si, Y.-W., & Leung, S. (2016). A hybrid algorithm based on variable neighbourhood for the strip packing problem. *Journal of Combinatorial Optimization, 32*(2), 513-530.

Zhang, D., Wei, L., Leung, S., & Chen, Q. (2013). A Binary Search Heuristic Algorithm Based on Randomized Local Search for the Rectangular Strip-Packing Problem. *INFORMS Journal on Computing, 25*(2), 332-345.

8 Appendix

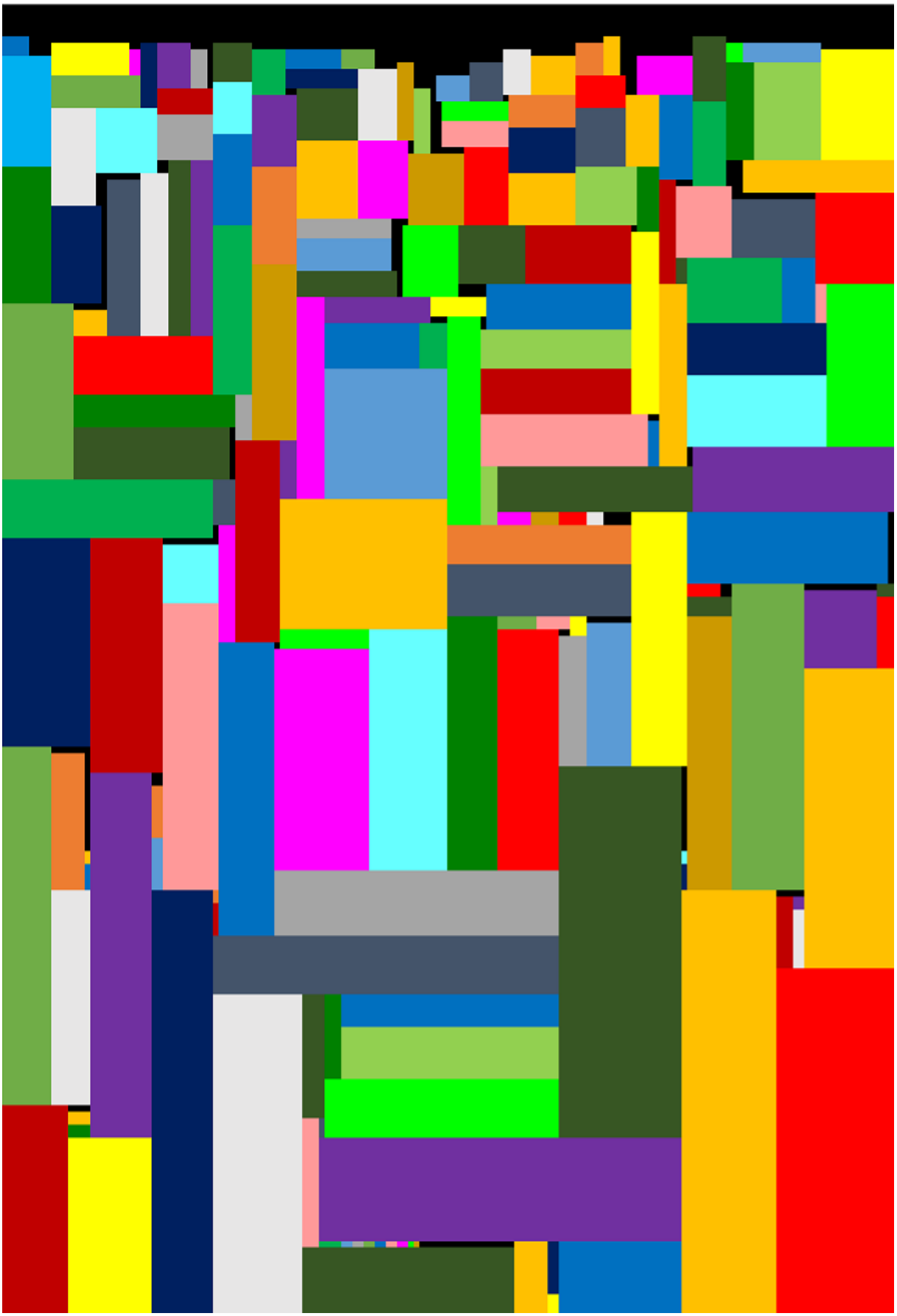

Figure 30: Sample result instance lw1972 from Hopper & Turton (2001)

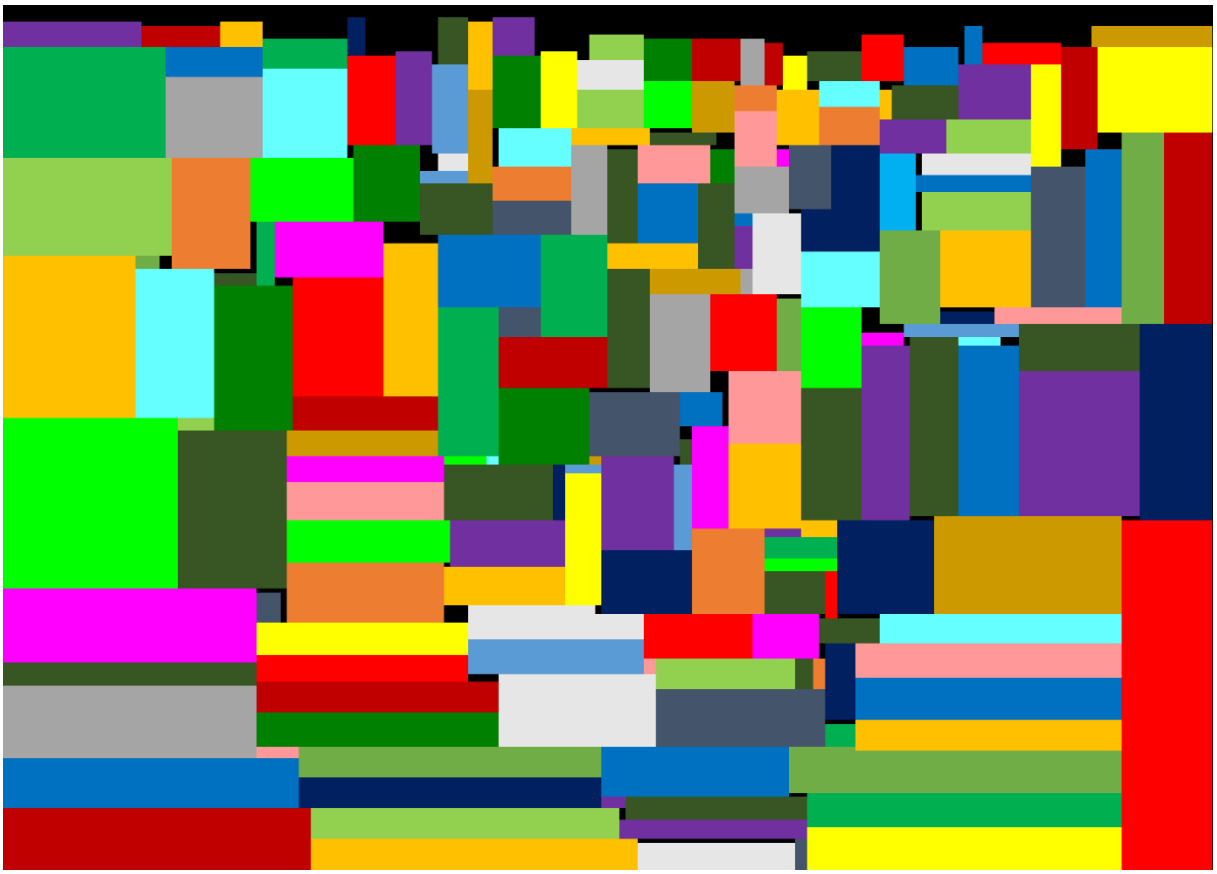

Figure 31: Sample results instance T7c from Hopper (2000)

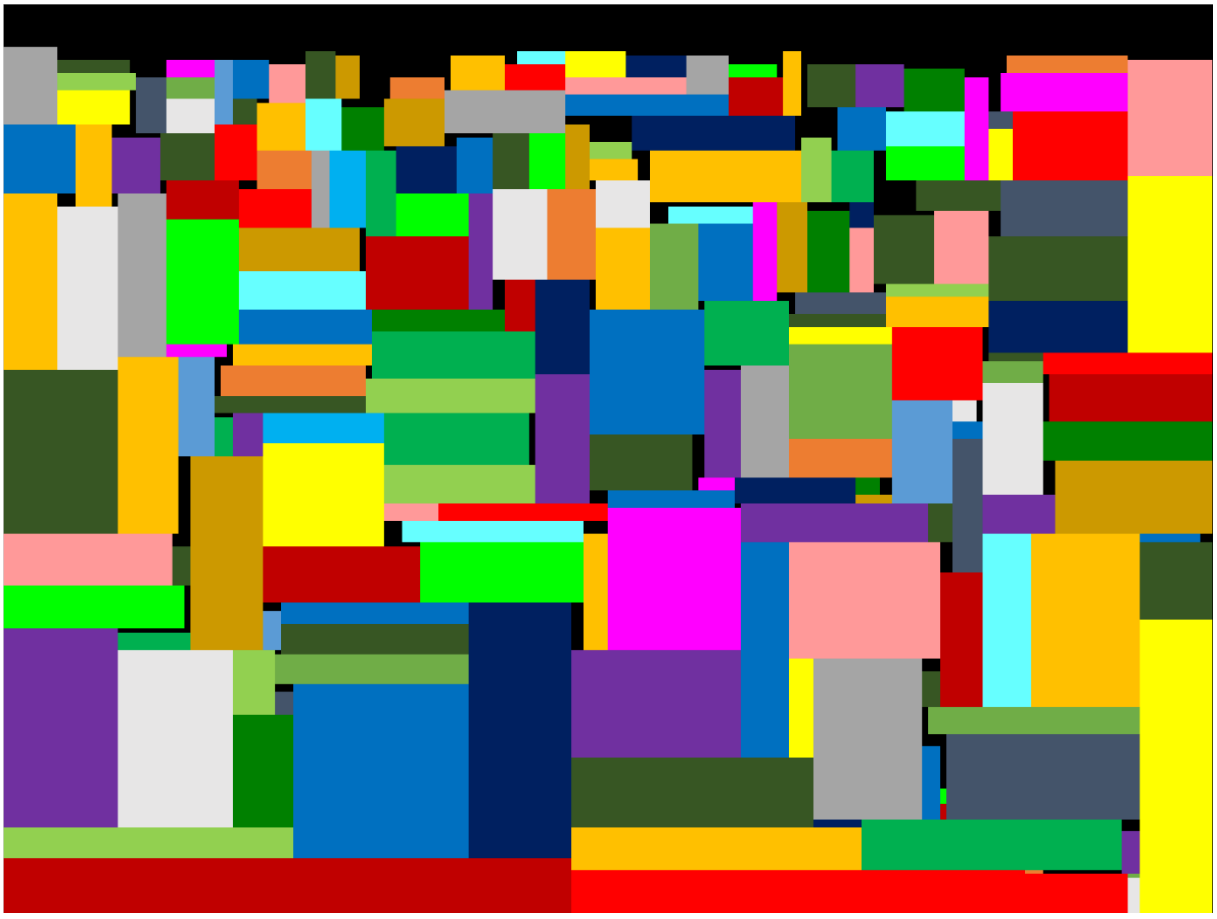

Figure 32: Sample result instance N7b from Hopper (2000)

Figure 33: Sample result instances N9 left and N12 right from Burke et al. (2004)

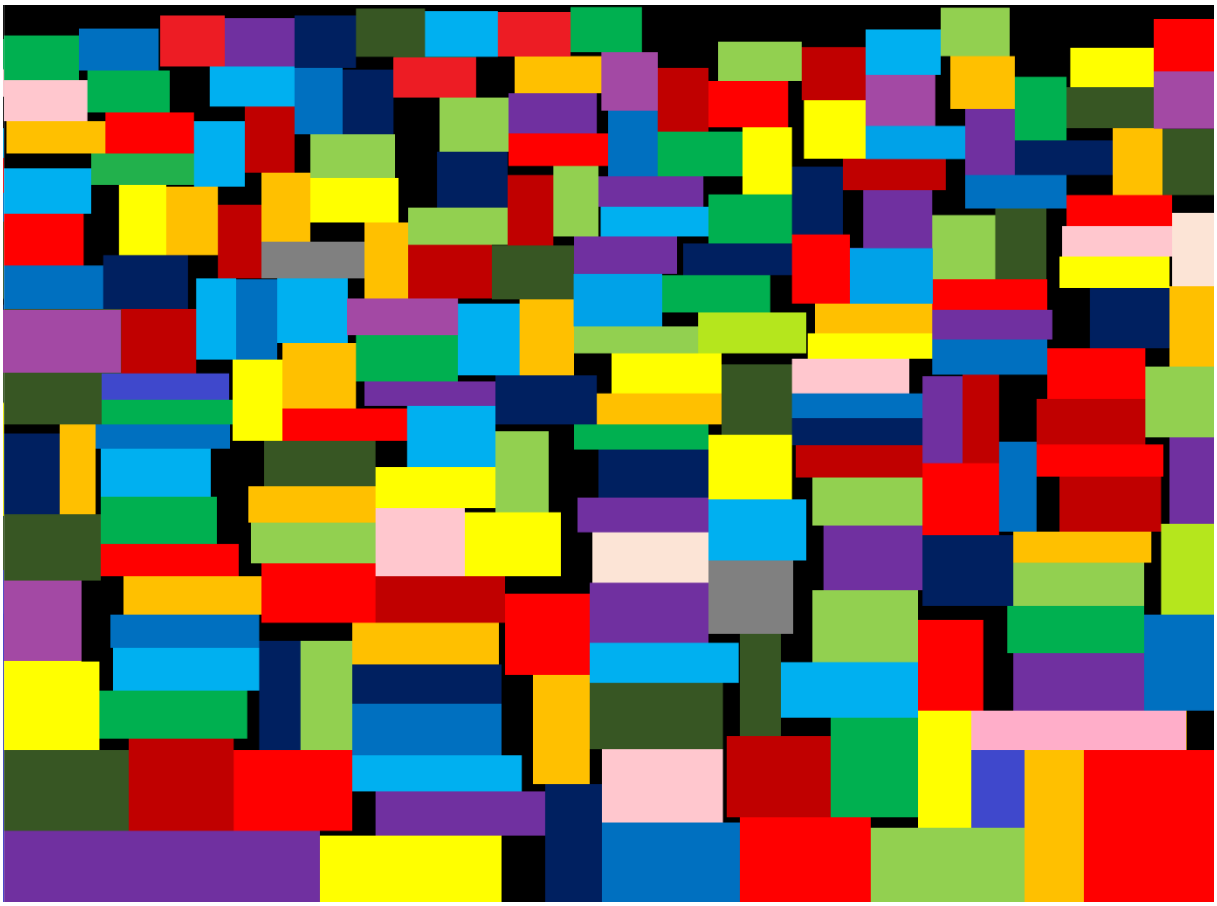

Figure 34: Sample result instance Nice4 from Wang & Valenzela (2001)

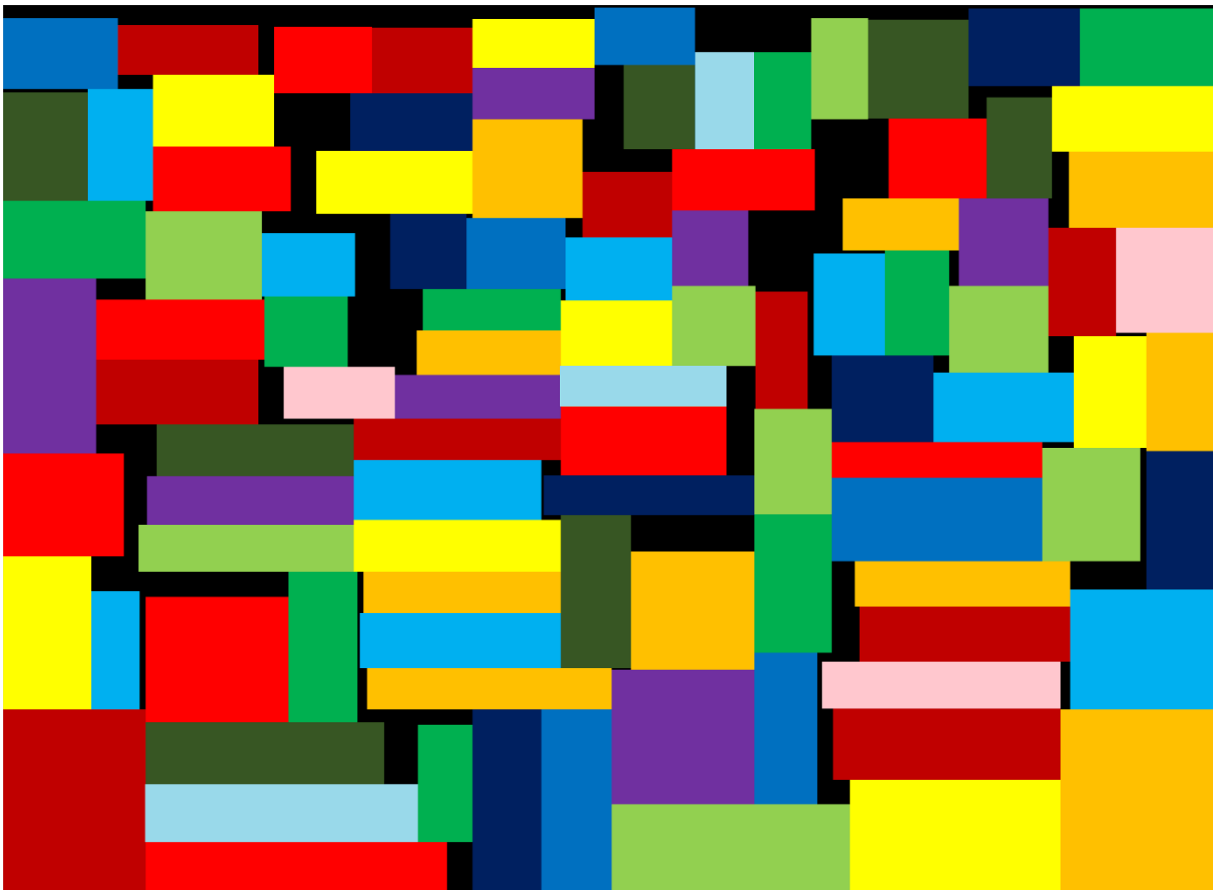

Figure 35: Sample result instance Nice3 from Wang & Valenzela (2001)

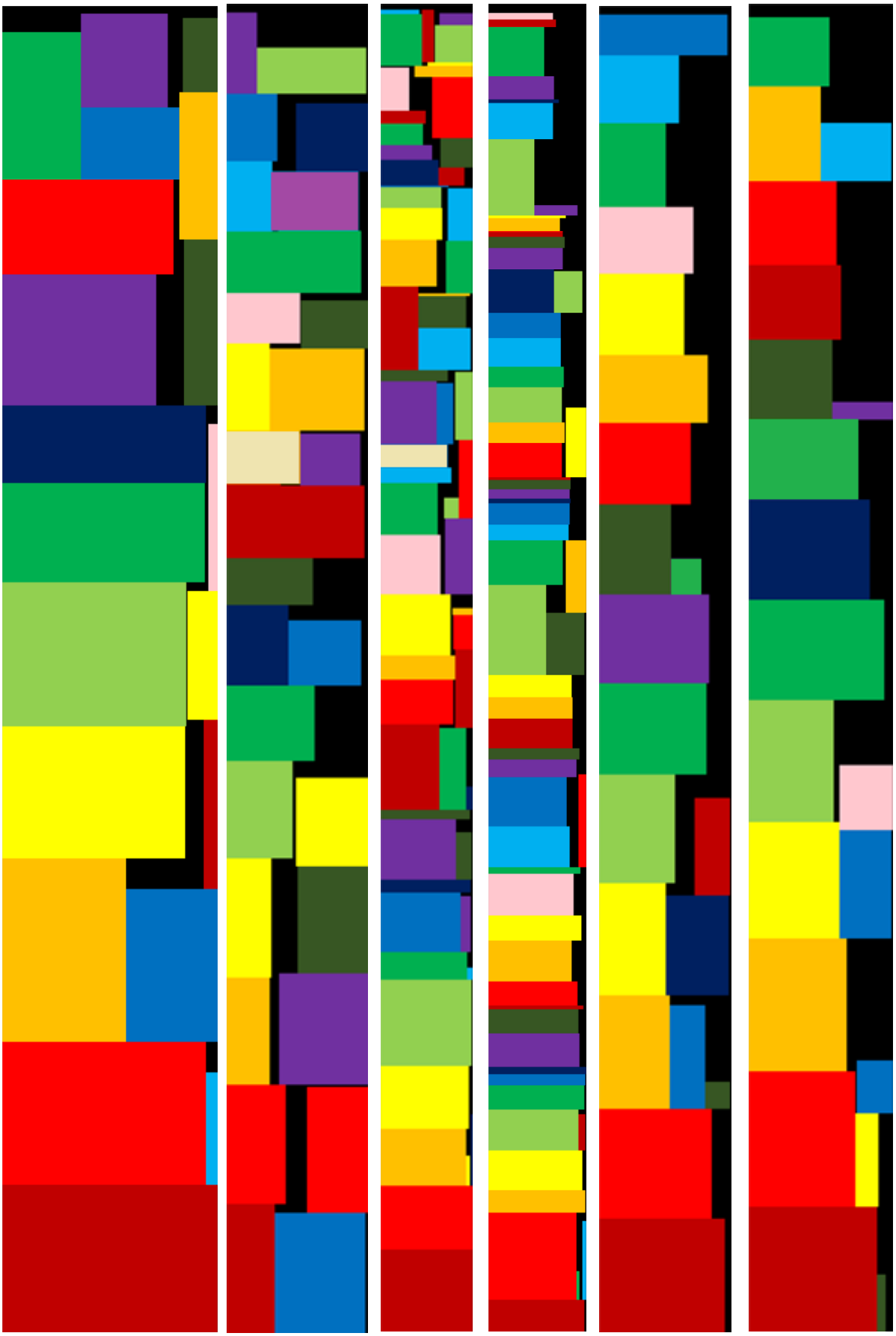

Figure 36: Sample results instances from left gcut02 and gcut03 from Beasley (1985a), C_5_229 from Berkey & Wang (1987) and C_7_325, C_9_408 and C_9_410 from Martello & Vigo (1998)