universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„New Extreme Points Packing Heuristics for the

2 Dimensional Strip Packing Problem"

verfasst von / submitted by

Baris Tandogan, BA

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

Master of Science (MSc)

Wien 2023/ Vienna 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 915
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Betriebswirtschaft
Betreut von / Supervisor:	Univ.-Prof. Mag. Dr. Karl Franz Dörner, Privatdoz.
Mitbetreut von / Co-Supervisor:	

Abstract

This thesis focuses on construction methods for the Orthogonal Packing Problems, particularly the two-dimensional strip packing problem. The Extreme Point Based Construction Heuristic, originally devised for bin packing problems, is adapted here for the two-dimensional strip packing problem. This thesis aims to demonstrate the unique characteristics of the Strip Packing Problem, showcasing its significant departure from solution dependencies on item sequence, unlike analogous problems such as vehicle routing problems. Through parameter optimization and experimental findings, the study can validate the detachment of solution quality from item sequencing and proposes opportunities to achieve high-quality solutions despite fixed sequences, crucial for real-life scenarios. While the focus remains on two dimensions, the research paves the way for potential extensions to problems involving additional dimensions and constraints. The implemented Extreme Point Heuristic proves itself to be effective, generating competitive solutions comparable to state-of-the-art methods, yet faces challenges in computational efficiency for larger instances. The conclusions highlight bottleneck characteristics hindering the algorithm's performance, offering directions for future research.

Zusammenfassung

Diese Arbeit konzentriert sich auf Konstruktionsmethoden für die orthogonalen Packungsprobleme, insbesondere das zweidimensionale strip packing Problem. Die Extreme Point basierte Konstruktionsheuristik, die ursprünglich für Bin-Packing-Probleme entwickelt wurde, wird hier für das zweidimensionale strip packing Problem angepasst. Das Ziel dieser Arbeit ist es, die einzigartigen Eigenschaften des Strip-Packing-Problems zu demonstrieren und seine deutliche Abkehr von Lösungsabhängigkeiten von der Artikelreihenfolge im Gegensatz zu ähnlich komplexen Problemen wie Vehicle Routing Problemen aufzuzeigen. Durch Parameteroptimierung und experimentelle Erkenntnisse kann die Studie die Trennung der Lösungsqualität von der Artikelsequenzierung validieren und Möglichkeiten vorschlagen, trotz fester Sequenzen qualitativ hochwertige Lösungen zu erzielen, die für reale Szenarien von hoher Bedeutung sind. Während der Schwerpunkt weiterhin auf zwei Dimensionen liegt, ebnet die Forschung den Weg für mögliche Erweiterungen auf Probleme mit zusätzlichen Dimensionen und Einschränkungen. Die implementierte Extreme Point Heuristic erweist sich als effektiv und generiert wettbewerbsfähige Lösungen, die mit modernsten Methoden vergleichbar sind, steht jedoch bei größeren Instanzen vor Herausforderungen hinsichtlich der Recheneffizienz. Die Schlussfolgerungen heben Engpassmerkmale hervor, die die Leistung des Algorithmus beeinträchtigen, und bieten Hinweise für zukünftige Forschung.

Contents

List of Abbreviations

NP-hard: non-deterministic polynomial hardness.

EP: Extreme Point Heuristic Crainic et al. 2007.

EPSP: Extreme Point Strip Pack Heuristic (this thesis).

BL: Bottom-up Left-justified; Bottom Left Heuristic Baker et al. 1980.

BLF: Bottom Left “F”; Bottom Left Fill Heuristic Chazelle 1983.

BF: Best Fit Heuristic Burke et al. 2004.

TP: Touching Perimeter Heuristic Lodi et al. 1999.

BBF: Bidirectional Best Fit Heuristic Özcan et al. 2009 and 2013.

GA: Genetic Algorithms Metaheuristic (implementation by Mumford Valenzuala et al. 2003).

GRASP: Greedy Randomized Search Procedure Metaheuristic (implementation by Alvarez-Valdez et al. 2008 [referred in the computational results], other implementations for 2DSPP in Beltran et al. 2004, Oviedo-Salas et al. 2022).

LS: Local Search.

SA: Simulated Annealing Metaheuristic (implementation by Burke et al. 2009).

SPP: Strip Packing Problem.

TSP: Travelling Salesman Problem.

VRP: Vehicle Routing Problem.

CVRP: Capacitated Vehicle Routing Problem.

HDA: Hybrid Demon Algorithm Metaheuristic Chen et al. 2005.

SRA: Simple Randomized Algorithm Metaheuristic Yang et al. 2013.

HSA: Hybrid Simulated Annealing Algorithm Leung et al. 2010.

HHA: Hierarchical search-embedded Hybrid Heuristic Algorithm Metaheuristic Chen et al. 2017.

SVC: SubKP single pass heuristic. Belov et al. 2008.

ISA: Intelligent Search Algorithm Metaheuristic Leung et al. 2011.

CIBA: Corner Increment Based Algorithm Metaheuristic Chen et al. 2018.

TSMSA: Local Search and Simulated Annealing added to CIBA Guo and Jiang 2022.

List of Figures

List of Tables

List of Algorithms

1 Introduction

Orthogonal Packing Problems are NP-hard combinatorial optimization problems where with the increase of dimensions and constraints the problem complexity increases in a nonlinear fashion. This makes the goal of finding optimal solutions increasingly hard and requires more problem-specific approaches with global objective-oriented improvements and modifications. Orthogonal Packing Problems lack a commonly used typology for the entire problem family, despite multiple papers defining variations, meaning there can occur confusion on what type of algorithm is implemented for better packing unless specified in very clear detail. The problem handled in this thesis, the two-dimensional strip packing problem, belongs to the same family of problems and is no exception to this. The most recent attempt to define a taxonomy for the Strip Packing Problem was by Oliveira et al. 2016 and Oliveria et al. 2022 where they present a diagram to show differences and what is commonly addressed in the literature.

This thesis implements a variation of the Extreme Point Based Construction Heuristic presented in Crainic et al. 2007 for the two-dimensional strip packing problem. The original heuristic was created for the bin packing problems (both for 2d and 3d). This is, at the time of writing, the first known implementation of this heuristic for the two-dimensional strip packing problem described in the following section. All of the implemented versions of heuristics will be explained in terms of typology as they cover all possible versions of a construction heuristic for two-dimensional packing problems that do not have guillotine cuts with the exception of the two-dimensional bin packing where the algorithms may differ in the initialization of bins in terms of how many bins they launch at once. In the later chapters, the parameters of this algorithm were tuned with a metaheuristic scheme and sensitivity of computational times were analyzed for different merit schemes. An algorithm ESPS|8| (Extreme Point Strip Pack 8) can create solutions comparable to other construction heuristics and when the parameters are fit well comparable to the strongest metaheuristics for the 2D Strip Packing Problem.

In this chapter, the problems is described and the literature on this problem is reviewed and summed up in a table. As the orthogonal packing problems are similar in nature, the solution methods can also often be interchangeably used. The literature in the review is limited to the strip packing problem but the references at the end of the thesis cover more problems than just the SPP. In chapter 2, the construction methods most relevant to this thesis are explained in detail. In chapter 3, the implementation of the heuristic is discussed in great detail and explained with pseudocode and the heuristic is further utilized in chapter 4 in combination with a metaheuristic scheme. Computational results are presented in chapter 5 and show that the implementation works well in comparison to other implementations but still has room for improvement. The thesis is concluded in chapter 6 with commentary on the results and further research opportunities on the problem.

1.1 Problem Description

The Strip Packing Problem typology is defined by Wäscher et al 2007 and this definition is further improved by Oliveira et al 2016, and Oliveira et al 2022 explains different approaches and the typology in finer detail. This problem has a single bin, with multiple possible dimensions, in which multiple shapes, often orthogonal need to be packed. Orthogonal means that the items are rectangular and the faces/edges of the items must be parallel to either one of the faces/edges of the bin which are perpendicular. The width of the bin is often defined and the height is allowed to be infinite. The goal of this problem is to pack all the items into the given bin while maintaining a minimum total strip height. Due to the objective of minimizing the strip height, in the implementation one can relax the infinite bin height to a reasonably high number above the best-known strip height to achieve results.

The strip height objective of this problem directly enforces compactness but in contrast to the bin packing problems the packing options are more limited as there exists only one strip/bin to pack into. In the bin packing problem where multiple bins might be opened at the same time, or the packing can be started from multiple corners of the bin to improve compactness, there exists more flexibility in terms of where and when to pack. The single bin with the unlimited height limits the flexibility but simultaneously increases sensitivity to parameters of packing, the most important being the sequence of items being input in case of online packing. This does not mean that the packing is only sensitive to the sequence of packages which will be further discussed in this thesis.

In the typology for this problem, there is often the mention of guillotine cuts which are straight line cuts made on the strip or part of the strip from one edge to another. Junior et al 2022 state that the guillotine cut solution methods are not addressed widely in the literature and after 2010 the number of solution methods with cuts seems to have decreased. This decrease is most likely due to the fact that guillotine cuts are often increasing complexity under circumstances where the items to be packed are not perfectly overlapping and might have irregularities similar to what may be found in real life. However, many instances that are commonly used are created with guillotine cuts, making it harder to actually understand how well an approach might work on different problem instances.

The Strip Packing Problem has many uses in real life, often being mentioned at the beginning of most papers as a key problem for stock cutting and furniture or palette packing as furniture and heavy appliances cannot be stacked when shipping. There are instances in the literature that simulate such scenarios due to their dimensions and their distribution which will be further discussed in this thesis. The guillotine cut instances and solution methods are interesting to read but with the advances in technology larger stock rolls, sheet metal, plywood, etc. are often cut with machines capable of moving in every possible axis, making any guillotine requirement obsolete. In the case of three-dimensional problems, guillotine cuts may be used for creating problem instance sets, but it is hard to justify guillotine cut constraints as any machinery that will be capable of

cutting and moving in three dimensions will most likely be capable of doing so without the guillotine cuts. Any machining involving a solid block, in this case, the bin/strip, will require adjustments after the process, such as uploading new models to a CNC miller and calibrating the machine for the next items, changing tips, etc. making any guillotine cut other than in one axis after milling objects sitting perpendicular to the cutting axis obsolete. This makes the three-dimensional guillotine cut only relevant in a particular theoretical setting, while the advances in technology make the two-dimensional guillotine cut only interesting in limited scenarios.

1.2 Literature Review

The above-mentioned sensitivity of the problem to the sequence of items inserted into the strip is the main focus of research in this field. The metaheuristics mentioned in this section use the sequence exclusively to improve the solution while some present different improvement and construction heuristics. In most cases, the construction heuristic and the improvement heuristic are problem specific and the metaheuristic can be more general in order to guide the heuristics in the search space.

The construction heuristic and the improvement heuristic are often the most challenging parts of such an implementation; therefore, many different approaches are present in the literature. Due to the similarity of approaches across multiple members of the OPP family, papers from the entire range were considered for this review.

Baker et al 1980, Chazelle 1983, Berkey Wang 1987, and Sheithauer 1999 present different approaches to the construction of solutions that are the bedrock of most metaheuristics in this problem domain. Lodi et al. 1999 present touching perimeter heuristic and other metaheuristic approaches, Beltran et al. 2004 implement a hybrid Grasp/VNS for this problem, Burke et al 2004 present the Best Fit construction heuristics, and Karabulut Inceoglu 2005 present Deepest BLF for the three-dimensional case. Bortfeldt 2005 implements a GA, Alvarez-Valdes 2006 implements a GRASP and Crainic et al 2007 implement the Extreme Point Heuristic for the 2SPP. Neveu et al. 2008 present an approach that focuses on holes in the packing, Imahori, Yagiura implement the best-fit heuristic in a very efficient way that can place millions of items, and Burke et al. 2009 improve their heuristic with a Simulated Annealing approach. Leung et al. 2010 implement another SA approach and compare this to GA, Allen et al. focus on the 3DSPP and implement an efficient version of Best Fit for this problem. Leung et al. 2011 present a fitness-based construction heuristic with a multi-start(sorting) LS and SA and Wei et al. 2011 present a skyline heuristic with different evaluation methods. Yang et al. 2012 have a construction heuristic based on wasted space similar to Leung et al. 2011 and a metaheuristic SRA. Da Silveria et al.2012 focus on the unloading constraints but the instances used are for the 2LCVRP. Özcan et al. 2009 and 2013 present the Bidirectional BF and a modified version that performs quite well for a standalone construction heuristic and Zhang et al. 2013 have a binary search algorithm with a corner point-based

construction that also focuses on wasted space. Cote et al. 2014 develop a Branch and Cut for the case with unloading constraints. Chen et al. 2015 create HDA that has a least waste-oriented construction based on Leung et al. 2011. Babaoglu 2017. Implements a Fruit Fly optimization, Lam et al. 2017 focus on GA for 2SPP, and Chen et al. 2019 use an approach quite similar to Extreme Points. Oviedo-Salas et al. also present a GRASP with a similar construction to Extreme Points with a skyline focus. Krebs et al. 2022 introduce an approach of subspaces in the bin for an increase in speed that was explored in this thesis. Que et al. 2023 are among a new group of papers that focus on deep reinforcement learning for the strip packing problems that present promising results. All the papers relevant for this thesis can be found in the bibliograpy. The papers that are relevant for 2SPP have been selected with additional details on methods and are presented below.

In this thesis, the Extreme Point construction heuristic from Crainic et al. 2007 was implemented for the 2DSPP with an alternative version of the fitness algorithm inspired by Wei et al 2018. Leung et al 2010, strongly influenced by the touching perimeter heuristic presented in Lodi et al. 1999. The spaces used to speed up the packing were influenced by Krebs et al. 2022 but were assigned to specific Extreme Points instead of objects. While some of these papers are not in the table below as they are not strictly Strip Packing literature, they contribute with ideas that can be implemented for this problems solution. The Best Fit and Bidirectional Best Fit heuristics were the strongest candidates to compare directly to the implemented heuristic and the Bottom Left and Bottom Left Fill results were included wherever available in Chapter 5: Computational Results.

Author(s)	Year	Method	Metaheuristic	Construction
Baker et al.	1980	Bottom Left	FALSE	TRUE
Chazelle	1983	Bottom Left Fill	FALSE	TRUE
Berkey, Wang	1987	bottom left, first fit, next fit, best strip	FALSE	TRUE
Scheithauer	1999	Corner Point	FALSE	TRUE
Lodi et al.	1999	Touching Perimeter	FALSE	TRUE
Mumford-Valenzuela et al.	2003	Genetic Algorithms	TRUE	FALSE
Beltran et al.	2004	Genetic Algorithms	TRUE	FALSE
Karabulut, Inceoglu	2004	Genetic Algorithms, Deepest BLF	TRUE	TRUE
Burke et al.	2004	Best Fit Heuristic	FALSE	TRUE
Bortfeldt	2005	Genetic Algorithms	TRUE	FALSE
Alvarez-Valdes	2006	Reactive GRASP	TRUE	FALSE
Crainic et al.	2007	Extreme Point Heuristic	FALSE	TRUE
Imahori, Yagiura	2009	Best Fit Heuristic	FALSE	TRUE
Önder,Ender	2009	Bidirectional Best Fit	FALSE	TRUE
Burke et al.	2009	Simulated Annealing, BF, BLF	TRUE	TRUE
Leung et al.	2010	Hybrid Simulated Annealing	TRUE	TRUE
Allen et al.	2010	3D Best Fit	FALSE	FALSE
Leung et al.	2011	ISA	TRUE	TRUE
Wei et al.	2011	Skyline, Iterative doubling binary search	TRUE	TRUE
da Silveria et al.	2012	multiple heursitics for SPP with unloading	TRUE	TRUE
Yang et al.	2013	LS,SA, least waste construction method with different fitness	TRUE	TRUE
Özcan, et al.	2013	Bidirectional Best Fit and modifications	FALSE	TRUE
Zhang et al.	2013	binary search (iterative local search with bounds)	TRUE	TRUE
Chen et al.	2015	HDA	TRUE	TRUE
Wei et al.	2016	improvements on the methods from Leung et al. 2011	TRUE	TRUE
Babaoglu	2017	Fruit Fly Algorithm	TRUE	FALSE
Lam et al.	2017	Genetic Algorithms	TRUE	FALSE
Chen et al.	2018	CIBA	TRUE	TRUE
Chen et al.	2019	best fit construction with global effect, extreme points	TRUE	TRUE
Rakotonirainy,Vuuren	2020	SA,GA, construction from Wei et al. 2016	TRUE	TRUE
Grandcolas, Pain-Barre	2021	Progress and Verify Strategy, novel local search	TRUE	TRUE
Guo,Jiang	2022	Local search, SA, complex constructions.	TRUE	TRUE
Oviedo-Salas et al.	2022	GRASP with EP(flags)	TRUE	TRUE
Que et al.	2023	deep reeinforcement learning, transformer	TRUE	TRUE

Table 1: Literature relevant for 2SPP

2 Construction Heuristics for Orthogonal Packing Problems

The Strip Packing Problem is unique where the packing can be done in many different ways completely disconnected from the sequence. This means that while the sequence is a big factor in solution quality it is rarely the only factor. There have been a handful of construction heuristics for this problem that have slightly different approaches to how to pack the items into the strip. The most relevant ones for this thesis were the Extreme Point Heuristic from Crainic et al. 2007, Best Fit from Burke et al. 2004, and Bidirectional Best Fit from Özcan et al. 2009. These heuristics were implemented in various papers and experimented with in different ways to increase efficiency such as in Imahori et al. 2009, Que et al. 2023, Chen et al. 2019, etc. In this chapter these heuristics are explained broadly with remarks on how the heuristic was later on implemented.

2.1 Extreme Points Heuristic

In this section, the original heuristic from the Crainic et al 2007 paper will be explained in detail as it is the basis of the heuristic made for the 2SPP in this thesis. The Extreme Point Heuristic of Crainic et al 2007 was heavily modified to be able to solve the 2SPP but the basic concepts have been adopted.

The Extreme Point approach is an extension of the Corner Point idea and comes from the implementation of shelf building in three-dimensional space. Basically, there is an envelope building that happens around the placed items with falling guillotine cuts. There are graph theory-based implementations as well as implementations with branch and bound for the corner point heuristic. The issue with the corner point heuristic is that it has a lot of wasted space as these cuts are made. Still, it reduces the computational time in contrast to other construction options as the placement is reduced to only a few possible points as opposed to sliding left and bottom. In Figure 1 one can see the corner point approach for the three-dimensional as well as two-dimensional space. The issue of the falling cuts is more obvious in the example of three dimensions and there exist differences in skyline approaches. The idea of corner points and them building this artificial wrapping around the existing packages is implemented also in Zhang et al. 2013 where the authors call this space envelope and all the gray space in the figure is this envelope space with the points as possible placement positions.

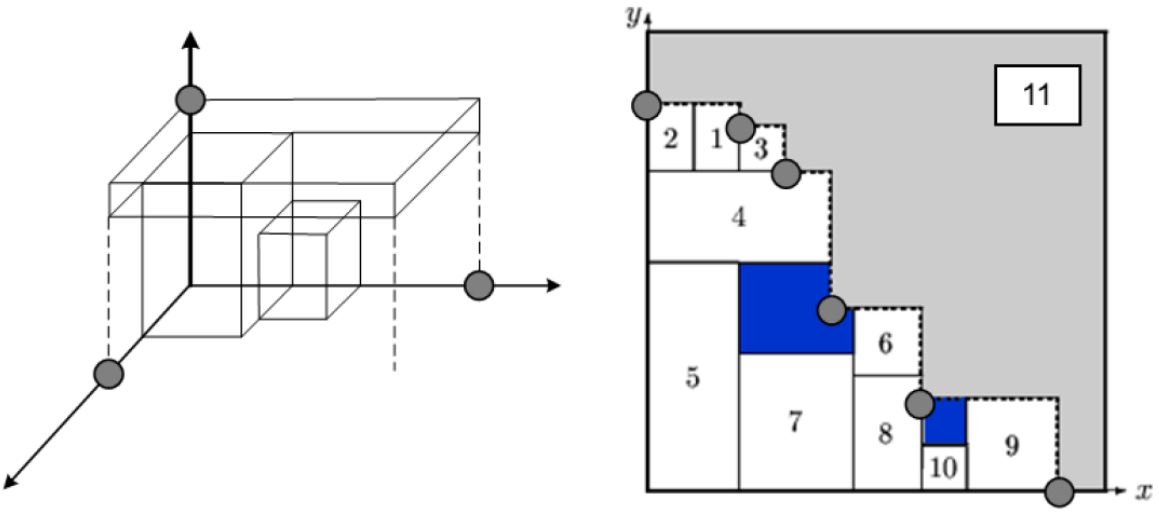


Figure 1 Crainic et al. 2007

The Extreme Point Heuristic is simply what Bottom Left Fill is to the Bottom Left heuristic. It includes more points for positioning that are beyond-the-envelope offerings. Yet all of the heuristics except the Best Fit based and some fitness-based skyline heuristics which will be explained in detail in the next section are adaptations or alternative implementations of the Bottom Left approach. In order to prove this theorem, this thesis offers the idea of handles and vertices for any future item that can be placed. Figure 2 shows the handles an item has ranging from 1 to 4 and any rotation does not affect the handle positions meaning the first handle is always on the bottom left side of a placed item. In the extreme point heuristic of Crainic et al., when placing an item into the bin or strip the focus always lies on where P1 may be placed. The corner point approach that can be seen in Figure 1 can only accept an item to be placed with P1 coinciding with that corner point. Alternatively, P4 could coincide with the corner of a hole, or additional corner points could exist on the right side to coincide with P2 or P3 but these cases are not relevant for these heuristics and will be considered in future chapters.

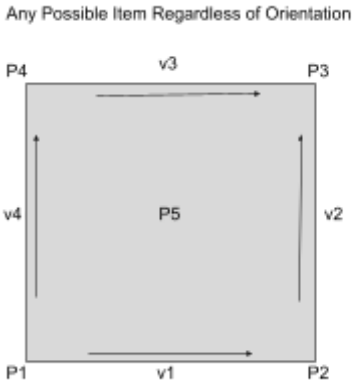


Figure 2

The Extreme Point Heuristic, in addition to the normal corner points of an envelope, has lower points with respect to their Y coordinate than the envelope allowed in the decision-making process

to place any item. The original heuristic has these additional points as well as some of the envelope points that do not correspond to a P4 or P2 of an already packed item not clearly defined which is a differentiation that will be made later on. The visualization of the extreme points can be seen in Figure 3 as it is presented in the original paper.

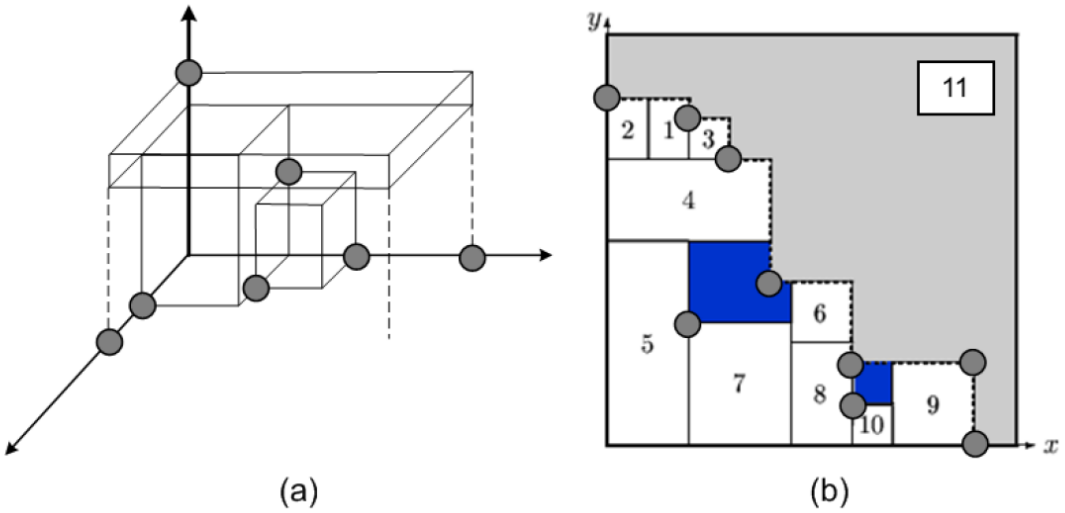


Figure 3 Crainic et al. 2007

The original heuristic is created for the bin packing problem and therefore there are two versions of the heuristic, namely the Firs Fit Decreasing and the Best Fit Decreasing. The former has the item loaded into the first bin it fits and where there is no possible placement a new bin is launched. The latter calculates all possible positions with a merit function and chooses the best position. The first fit option is clearly the faster option but the best fit should construct better results given the merit function represents the objective well.

The original implementation of the heuristic considers multiple ways of ordering the items, simply preprocessing depending on criteria such as decreasing volume, height, and area (in the 3D case for the base, which would correspond to the width for the 2D problems). Additionally, multiple merit functions were tested where the values were depending on residual space in the bin, level packing, and residual space for the extreme points meaning the future packing space corresponding to the remaining extreme points. While the different alternatives of merit functions might make sense for the bin packing problem, the strip packing problem has infinite height, and the idea of residual space for the extreme points cannot be implemented without setting artificial boundaries and the preference for level packing might cause overall worse solutions. The focus of the paper lies more on the preprocessing of items than details of merit functions and which ones were used for which results and the shown results compare the sorting criteria only. The extreme point heuristic performs unsurprisingly better than the corner point heuristic as it simply allows for more placement options in the bin.

2.2 Bottom Left and Bottom Left Fill

Bottom-up Left-justified (BL) heuristics were first defined by Baker et al. 1980. In the paper, the bottom left packing is explained by pushing the next item to the lowest possible position from above (skyline) and then to the leftmost position it can be moved towards. The paper mostly focuses on different scenarios with their quality and what happens when the items are packed by width and height and it is mentioned that preprocessing the items by width is a good idea.
Chazelle 1983 introduces BLF which is what is addressed as Bottom Left Fill nowadays but there is no mention of the “Fill” in the paper but rather the data structure implemented for the holes called “F”. The paper focuses very strictly on the holes, how they should be structured and filled, how and which vertices to find and define the holes with. The Extreme Point Heuristic can be considered a modified BLF approach with a fitness function as instead of forcing the most bottom and most left position the heuristic considers multiple positions and chooses the best one. This approach appears significantly different from BLF but the items are always placed with their corner point where their bottom and left vertices overlap, as is done in BL and BLF. The way the items are placed the same, the positions where they are placed vary greatly.

2.3 Best Fit and Bidirectional Best Fit

The Best Fit Heuristic introduced by Burke et al 2004 adds some new ideas to basic packing heuristics. This is also the first paper where the idea of extreme points and how the possible positions can be stored for the BLF are proposed as can be seen in Figure 4.

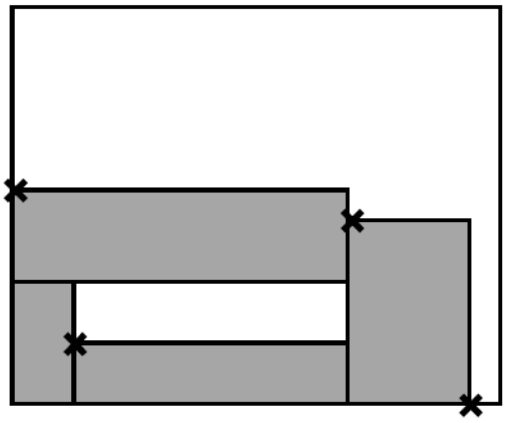


Figure 4 Burke et al. 2004

The paper also mentioned the advantage of non-guillotine solution methods outperforming guillotine ones with some computational disadvantages. The Best Fit Heuristic has a clear advantage over the BL and BLF as it chooses the position to pack and the best item to pack for this position. This approach is very similar to how the deep reinforcement learning implemented in

Que et al. 2023 reacts in order to pack the items. The lowest position is chosen and then the best item for this lowest position is picked from the remaining items. If the given niche placement policy cannot pack an item into the given niche the corresponding niche is raised to the next lowest possible niche, expanding its width. All the considered niches here are skyline niches meaning when an item is packed into it it must always have the upper vertex, previously defined as v3 in Figure 2 completely exposed to access from the upper opening of the strip. Raising the niches can be done with various policies here including the minimum dimension constraint pushing it higher if a package with the minimum dimension could not fit in it. The interesting detail about the Best Fit heuristic is that it is only skyline based meaning there is a high sensitivity when it comes to packing sequence, but this sensitivity is lower than BL and BLF approaches as the heuristic chooses first the niche then the item, overriding whichever packing sequence is input at each iteration. Improvement heuristics on the BF and BBF therefore must have some kind of chunk swap where the sequence is swapped with not individual items but with sections of the sequence. Additionally, the initial quartile of the sequence carries a much higher importance with heuristics where the sequence of input may be overridden as it creates the initial niches that are the skyline on the bottom section that then get filled further with a method that has a lower sensitivity to the remainder of the sequence.

Very interesting would be a sensitivity analysis of sequence and parts of the sequence of all construction heuristics as it would give a clear guideline on what kind of search operators to implement in order to improve solutions. The BL and BLF approaches have a higher sensitivity to the input sequence with BL clearly leading as it is forced to the lowest possible niche on the skyline with the next item and the BLF is forced to the lowest possible niche on the entire strip with the next item.

The advantage of the BF over BL is that the items are placed with policies, these policies are placement at the leftmost, at the tallest, or the shortest neighbor. To clearly define the meaning of this advantage: items can be placed also on the right side instead of the left side, a feature introduced by the BF and then used in various ways by other heuristics. A merit function here could be implemented similarly to what is done in the Extreme Point Heuristic or the proposed fitness function of the Extreme Point Strip Pack (EPSP) heuristic in this thesis. A simplification of the skyline problem is that it is saved as an array of the length W and saves the highest item at each coordinate in this array. If the placement is done at a specific niche, the packing problem simply becomes a basic summation of numbers, and any fitness function would become simple maths operations. In the paper, the placement policies are mentioned for specific results, very similar to what this thesis has to offer regarding what preferences were used for packing in a more numerically specific way.

The Bidirectional Best Fit Heuristic introduced by Özcan et al. 2009 and 2013 also considers vertical niches. These vertical niches are niches created on the left side with the theoretical lower bound of the strip (using this theoretical skyline vertex as a neighbor. There are various placement

policies added and introduced in the paper and additionally, a modified version is presented where there is a preprocessing done without a strip/bin where the items are connected to each other with their same or similar sizes vertices. In this case, if two items have the chosen vertices connected creating a larger item that can accommodate more items connected to it. This is also done with different policies to create this level meta item that can be placed nicely into the strip. This increases computational time substantially but creates very good results.

Both the BF and BBF benefit from guillotine cut instances as the placements are preferring perfect overlaps. As the implementations of both were tested on the available instances at the time, it is impossible to predict how they would perform on different datasets that can be created in the future without reimplementing the structures and policies identically. The benefits of the heuristics for specific instances can be observed after becoming familiar with the instances, yet this benefit can only be speculation without comparing all heuristics and stress-testing them with datasets created for the purpose. Although they are the building blocks for most metaheuristic implementations for solving SPP and BPP, at the time of writing this thesis, most of the construction heuristics including BL, BLF, BF, BBF, and Toching Perimeter TP are largely untested in terms of their sensitivity to parameters, sequences, policies, item dimensions, and distributions of dimensions and irregularities. While most of them provide some theoretical time complexity notation, these only remain theoretical. A survey on all construction methods for the OPP family with intensive comparisons still remains to be desired. The relevance of such research is that given a dataset or a specific problem, some heuristics and their use can be ruled out. This would not mean that they work worse in general but are just not suitable for the given problem but perhaps superior or faster with another set of problems. It could simply mean that a construction heuristic metaheuristic pair can reach a better result than another one could not in a short time for the specific problem instance.

3 The Extreme Point Strip Pack Heuristic

The implementation of the Extrem Point Heuristic for the 2DSPP (EPSP) presented multiple issues from the get-go. In contrast to the Bin Packing Problem, SPP has the potential to create a lot more Extreme Points as the bin is infinitely large, and sometimes the number of items can be quite high. In this section, this thesis will go into detail on the implementation, explain different versions of the algorithm with the adopted typology, and depict the data structures chosen.

During the implementation of any construction heuristic, the main objective is to have it run as fast as possible while delivering good enough results. The reason for this is that most of the optimal or benchmark solutions can only be achieved by exploring different parts of the total search space and each iteration of a solution needs to be constructed, improved, and saved on some acceptance criteria. The more iterations of this process can be repeated for, the higher the

chances that a larger section of the search space is covered. For the 2DSPP and most of the OPP family, the search space that can be created by different heuristics is infinitely larger as not only the sequence of items but their relation to each other and to the bin/strip can vary.
Each construction heuristic creates a neighborhood for the improvement heuristic to work on. This makes both of the heuristics extremely problem specific and similar to any other problem the higher number of total runs means a higher number of new possible solutions and more modifications on the solutions. For the 2DSPP the construction heuristic almost singularly decides the faith of the quality of the final solution given the improvement heuristic can utilize its strengths well. The above-mentioned requirement for speed and quality from the construction heuristic is so that the building of the solution does not bottleneck the entire process of diversification in the neighborhood that can be created by the operators. The speed here does not only mean how fast a given instance runs but also how well the algorithm scales and how the parameters affect its speed on the given instance. The quality can mean the objective value but also how prone to improve the solutions are given the improvement operator can work on the solution effectively.
When implementing the heuristic and testing for the instances all of these were factored in which will be explained further in detail in upcoming sections. For various ways of packing a strip without guillotine cuts 12 different algorithms were identified and implemented. For the typology of these algorithms, the reader can refer to Füllerer et al. 2007 for their explanation of the classification of loading configurations. The classification adopted here has additional details on how the input is given and how the input is dealt with prior to loading.

Algorithm Family Extreme Point Strip Pack (EPSP):

1. On-|UO|: online input(the input sequence may not be modified), unrestricted (no sequentiality, unloading constraint) placement, oriented (no rotation) loading.
2. On-|SO|: online input, sequential placement, oriented loading.
3. On-|UR|: online input, unrestricted placement, non-oriented (no restriction on rotation) loading.
4. On-|SR|: online input, sequential placement, non-oriented loading.
5. On-|UR|-PreR: online input, unrestricted placement, pre-rotation of items prior to loading.
6. On-|SR|-PreR: online input, sequential placement, pre-rotation of items prior to loading.
7. Off-|UO|: offline input, unrestricted placement, oriented loading.
8. Off-|UR|: offline input, unrestricted placement, non-oriented loading.
9. Off-|UR|-PreR: offline input, unrestricted placement, pre-rotation of items prior to loading.
10. Off-|UR|-Prep: offline input, unrestricted placement, non-oriented loading, pre-processing of items prior to loading.
11. Off-|UO|-Prep: offline input, unrestricted placement, oriented loading, pre-processing of items prior to loading.

12. Off-|UR|-PrepPreR: offline input, unrestricted placement, pre-processing of items prior to loading, pre-rotation of items prior to loading.

One main question that arises when comparing these is what the online/offline input notation means in contrast to the sequential/unrestricted placement notation. The sequentiality of the placement means that the items are placed only on the skyline, which is what BL, BF, and BBF offer. The items placed in their sequence should be removable from the strip in the given sequence as well. Whether the input is online or offline means that the next item to be placed anywhere is fixed or can be chosen from any item. The way offline algorithms view the input sequence can be decided to have a limited length, a Restricted Candidate List RCL like Greedy Randomized Adaptive Search Procedures (GRASP) have to offer, or totally unlimited with the number of remaining items put into consideration for the placement. Algorithm 8, Off-|UR|, adopts the latter and calculates each remaining item for each possible placement at each iteration. This does not scale well and will be explained in further detail when presenting how the number of Extreme Points scales with increasing item size and diversity of item dimensions.

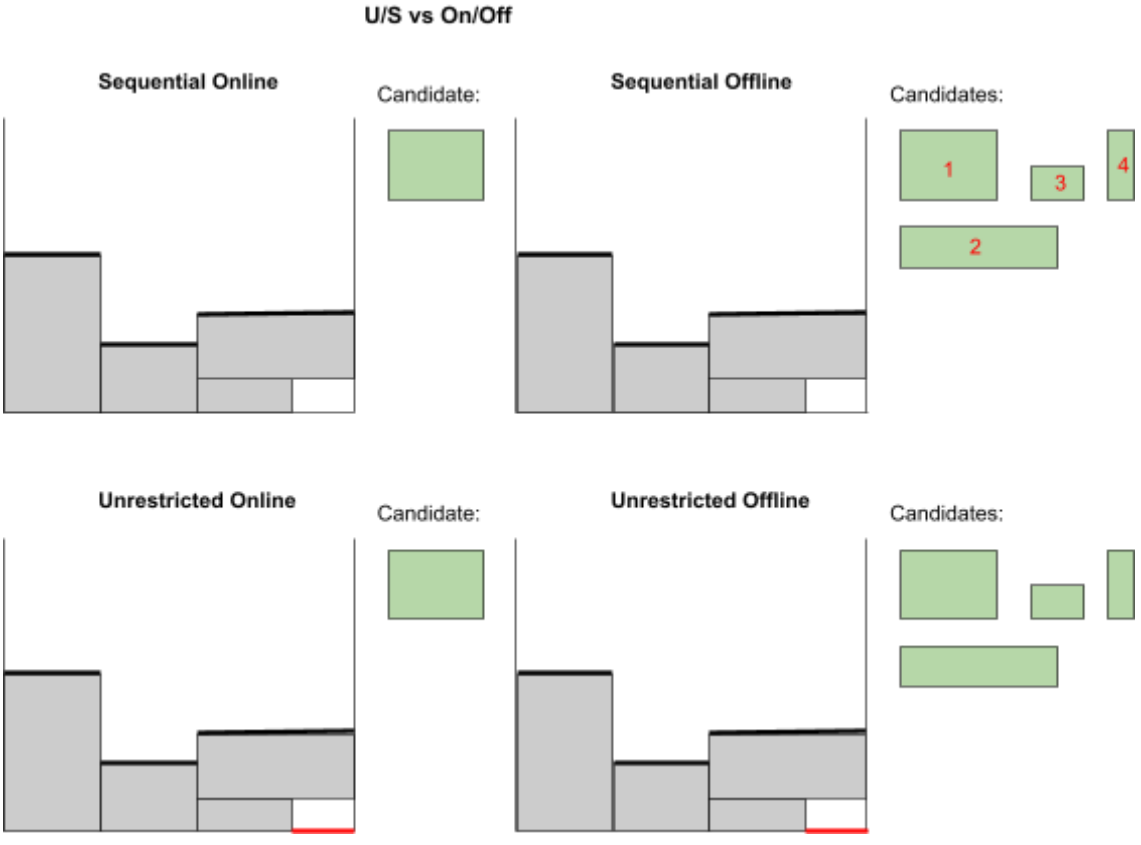


Figure 5

In Figure 5, one can see the differences between sequential loading constraints versus online and offline input constraints. In a sequential online case, there is only the skyline available for loading as the item needs to be fully available to unload when the number of items remaining to unload is at this given point. In the offline sequential case, there are more items to consider at the same time

but they must respect each other as the sequence matters, they create a new skyline with each possible placement so with a given order of 4 packages, each placement fitness must be calculated with respect to how it affects the skyline for the other items and in consequence the objective value and newly created holes. The sequential offline case is not necessarily the simplest case to implement as with the increasing number of candidates the number of calculations increases exponentially. This case is very similar to what is mentioned in the Chen et al. 2019 paper where the fitness of an item at a position is calculated with a finished packing of all of the remaining items after the placement is done. In this manner, the approach is more global as each fitness value has the objective value considered in it but with an increasing number of items to consider and pack the approach becomes infeasible to implement to work within reasonable computational times. The unrestricted online and offline cases have the red-marked bottom section as well as the skyline to consider for placement. The offline case can consider multiple items for placement whereas the online case can only consider one item per iteration. An offline preprocessed algorithm would also consider only one item at a time after sorting the items prior to loading according to some criteria such as decreasing width, decreasing volume, decreasing height, etc. giving it control over the sequence without increasing the computational time exponentially.

The two main algorithms focused on in this thesis were algorithm 8, Off-|UR|, and algorithm 10, Off-|UR|-Prep. The former considers each item at each possible placement at each iteration, giving it the right to dynamically override the input sequence. On the other hand, the latter considers the next item for all possible placement positions after preprocessing all items in the beginning according to some criteria. While in theory the former should provide much better solutions as it can freely decide a pair of item and position at each iteration, due to the number of calculations it has to do at each iteration it could not be considered for bigger problems and as it overrides the sequence dynamically it cannot be considered for any metaheuristic approach on the sequence. The latter, while it might seem like it also has control over sequence and therefore is not fit for metaheuristics, is the basis for algorithm 3, On-|UR|, where the sequence cannot be overridden. It makes sense to start the sequence somewhere and sorting according to a decreasing parameter seems to be the default in all heuristics and metaheuristics.

Any optimization on the sequence was omitted from this thesis as the focus lies strictly on the construction and comparison to other construction heuristics. Nevertheless, the sequence is the most important parameter of such a heuristic. The observation made during the implementation was that while the sequence of input is important, there are too often cases where subsequences are obsolete. An example can be seen in Figure 6 where given a fitness function where the items are not forced to the lowest possible placement but rather only prioritize touching perimeter and perfect overlaps, the subsequence of items 1-4 that might be placed on the skyline is totally obsolete. This means that the input sequence can be [1,2,3,4], [4,3,2,1], or any other possible version of this chain, but the solution would still be the same as they do not affect placement

possibilities for each other with the given fitness function. While this is a small example, in a case where there are over a thousand Extreme Points to place these 4 items, the problem becomes more obvious. Any improvement heuristic on the sequence needs to be able to identify where an item placement makes a global difference and work on improving either the specific item placement or which other item to replace at the position. This obsolescence of subsequences renders the development of an improvement heuristic highly problem specific and complicated. Especially given that the runtime might not be particularly short, it becomes challenging to develop an operator that can promise the steepest descent or any descent at all by swapping elements in a sequence.

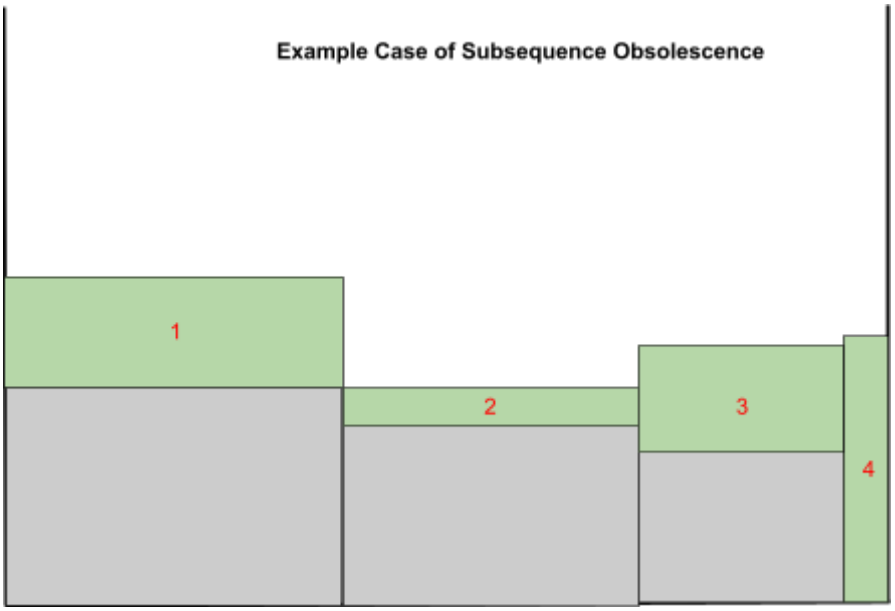


Figure 6

3.1 Data Structures

For the implementation of the heuristic, the decision on the data structures was a key consideration. In the end, the items were defined as in Figure 2 with four corner points and four vertices. Each vertex has its own Point 1 and Point 2 where the latter always has a higher coordinate on one axis so that the former is either on the left or under the latter when viewed on the coordinate system.

Unlike other papers such as Burke et al. 2004, no arrays of matrixes were used for the strip to see where it is occupied and where there is free space. Upon packing an item, the candidate's extreme points are added to a list of active extreme points. The other points are not added anywhere as they do not represent possible placement points. The four vertices of the item are to be considered next. First, the algorithm checks whether there are any vertices (from a list of vertices) that are coincident with the distinct vertex. If there are any collinear coincident (moving on the same axis and touching each other) vertices they are merged to a larger vertex. If this is not the case the

vertex is added by itself to the list. This structure helps build walls and levels as it forces the merge. When this merge happens, unlike other packing procedures like in Wei et al. 2018 where the level packing is incentivized as it reduces the number of future possible packing positions, the extreme points on this longer merged vertex are not removed. This causes an increase of calculations that need to be done at each iteration but as the algorithm cannot slide items around (therefore move future items independently from the dimensions of the already packed items in the strip) this proved to create higher quality solutions as opposed to neglecting the possible placement points on merged longer vertices.

Another definition that needed to be made was the extreme points that arise from the virtual extension of some vertices. This can be clearly seen in Figure 3 from Crainic et al. 2007 where items 4 and 6 and items 8 and 9 create possible placement points that do not inherently exist due to the items but due to their specific placement in the strip. The points that are extending from a single vertex onto another existing but not extending vertex were defined as shadow points in this implementation. Additionally, the points that are extending from two vertices and do not rest on an existing vertex were defined as float points. When two vertices may extend to create shadow points and they cross over each other, this creates float points. In Figure 7 the difference can be observed clearly.

For the shadow and float points a maximum search allowance needed to be defined. This search idea was repeated in other ways as well in the algorithm and for this, the size of 2*Maximum dimension was chosen where the maximum dimension is the largest dimension of the input items. This maximum dimension is found before the packing with an analysis of the input file. Beyond this, the creation of shadow points so far away was not beneficial because the shadow points can be cast on multiple items (e.g. a new item is packed into the strip, overlapping with a virtual vertex that was extending from an extreme point onto a shadow point) increasing the number of possible packing positions rapidly with no observed improvement to the solution quality. When these shadow points were allowed to extend so far, they were creating more possible packing positions simply for the sake of creating them and not because, within a few iterations, the existence of that shadow point would provide a good overlap with the vertex that has cast that shadow.

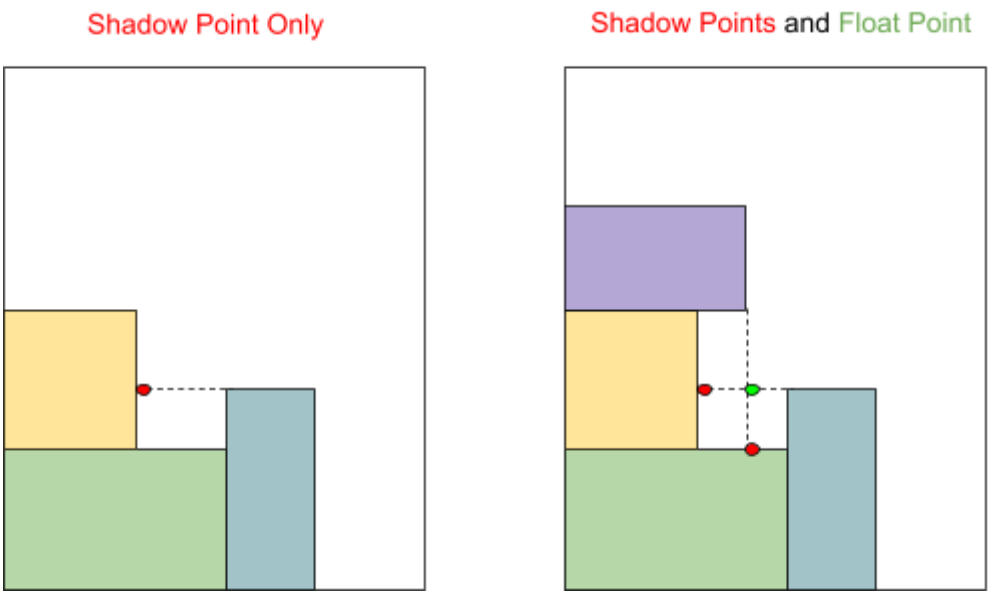


Figure 7

Algorithm 1 is used to refresh the active extreme points at each iteration of packing an item.

Algorithm 1: Refresh Extreme Points

(Package Chosen_item, ExtremePoint Chosen_Epoint)

```
1   P2,P4 ← Chosen_item.P2,P4
2   V2,V3 ← Chosen_item.V2,V3
3   VerticesTouchingP2_V,VerticesTouchingP2_H  ← CALL Fetchcoincident_vertices(Activevertices, P2, Vertical,Horizontal)
4   VerticesTouchingP4_V,VerticesTouchingP4_H  ← CALL Fetchcoincident_vertices(Activevertices, P4, Vertical,Horizontal)
5
6   IF (chosen Extreme Point and Item increase the Stripheight)
7           Stripheight ← Chosen_Epoint.Y + Chosen_item.Length
8   ENDIF
9   ActiveExtremePoints.Remove(Chosen_Epoint)
10
11  IF ((P2 is a viable option for inserting the smallest item at) AND (P2 is allowed))
12          ActiveExtremePoints.Add(P2)
13  ENDIF
14
15  IF (VerticesTouchingP2_V.Count EQUALS 0)
16          IF (VerticesTouchingP2_H.Count EQUALS 0)   //effectively no existing vertices touch P2
17                  Shadowvertical ← Create a new shadow point dropping down from P2
18                  viable? ← CALL Overlapcheck(Shadowvertical)
19                  IF (the point is viable)
20                          ActiveExtremePoints.Add(Shadowvertical)
21                          VirtualVertices.add(Shadowvertical.Virtualvertex
22                  ENDIF
23          ENDIF
24  ELSE
25          CALL MergeVertices(VerticesTouchingP2_V, V2)
```

```
26
27  IF ((P4 is a viable option for inserting the smallest item at) AND (P4 is allowed))
28          ActiveExtremePoints.Add(P4)
29  ENDIF
30
31  IF (VerticesTouchingP4_H.Count EQUALS 0)
32          IF(VerticesTouchingP4_V.Count EQUALS 0) //effectively no existing vertices touch P4
33                  Shadowhorizontal ← Create a new shadow point moving left from P4
34                  viable? ← Overlapcheck(Shadowhorizontal)
35                  IF(the point is viable)
36                          ActiveExtremePoints.Add(Shadowhorizontal)
37                          VirtualVertices.add(Shadowhorizontal.Virtualvertex)
38                  ENDIF
39          ENDIF
40  ELSE
41          MergeVertices(VerticesTouchingP4, V3)
42  ENDIF
43
44  FOR EACH (Vertex V in VirtualVertices)
45          colliding? ← CALL CheckCrossover(V)
46          IF (the virtual vertices are colliding)
47                  NewShadowPoints = CALL IdentifyShadowPoints(V)
48          ENDIF
49  END FOR EACH
50  FOR EACH (Extreme Point E in ActiveExtremePoints)
51          viable? ← CALL Overlapcheck(E)
52          IF (the point is viable)
53                  ActiveExtremePoints.Remove(E)
54          ENDIF
55  END FOR EACH
56  RETURN
```

The vertices V1 and V4 of the chosen item are handled in another algorithm to refresh the vertices, remove duplicates, and identify overlaps.

The implementation also has limited the search for the vertices, points, and items for defining other subspaces of the extreme points. When an item is placed, only the nearby extreme points were updated with a given search space multiplier (multiplying the maximum dimension). Each Extreme Point has a list of rules that represent Vertices saved inside them that defines the nearby space. This space idea was presented by Krebs et al.2022 but modified so that individual points have spaces that belong to them. If a point of an item that is going to enter this space after positioning its corner point to the extreme point exceeds and breaks a rule, automatically the item is not allowed to be positioned in this way and no further fitness is calculated for it. This movement and breaking of rules can be seen in Figure 8.

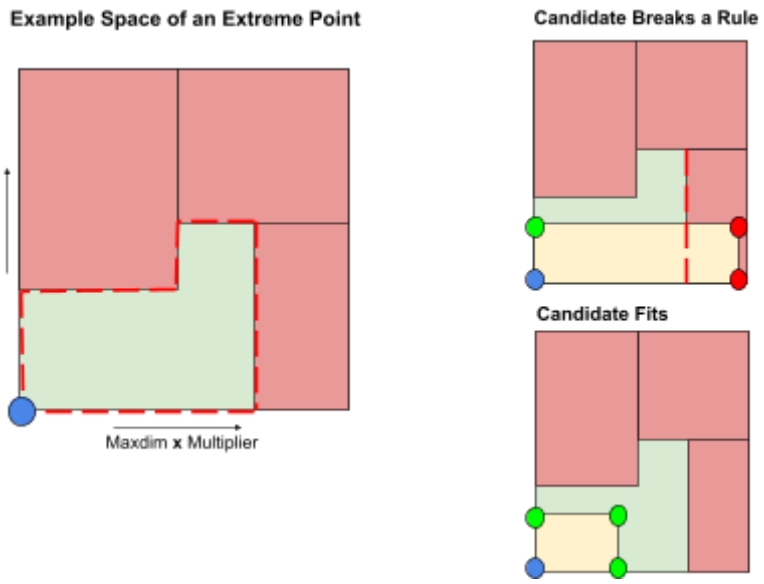


Figure 8

The frequency of how often these spaces are updated depends on the proximity of the last placed item to the strip. The overlap check which eliminates existing Extreme Points also works similarly with this proximity policy. If a new item is placed in close proximity to the Extreme Point, the space of this point is updated. This works exactly as the placement worked in Figure 8 but this time instead of checking for overlaps with any other items, the algorithm simply checks whether the given item with its already decided coordinates (as it is the last placed item) falls in the borders of the space of the considered Extreme Point. If it does, then the space of this Extreme Point needs refreshing. As this refreshing process requires fetching numerous coincident vertices it is time-consuming and is avoided this way when it is not required.

Similarly for checking overlaps, all items in close proximity to this Extreme Point are retrieved, and multiple new points offsetting from the Extreme Point are tested for breaking any rules on the vertices of this item. These new points also include points to check for the minimum dimension as if an item with the minimum dimension cannot be placed onto this Extreme Point, it need not be considered further for calculations. The reason for also considering a 1x1 Dummy is that sometimes in the implementation the minimum dimension is updated and the case can occur where a very item was placed very close to an Extreme Point, the minimum dimension was updated so that the new Dummy minimum dimension item falls only into an allowed position with its corner points although it would actually not fit into the space if it would be updated and the item would be tried for this position. This is one of the disadvantages of the structure that was implemented here as it only works with corner points of items and edge points of vertices and not the range or the area between those.

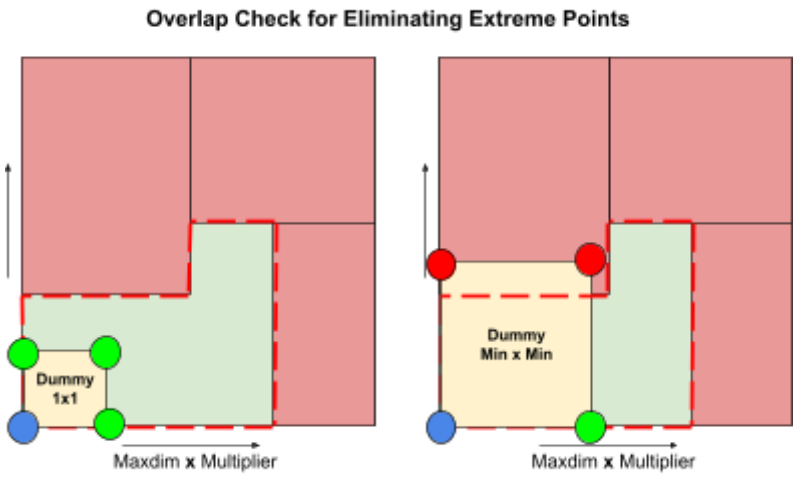


Figure 9

3.2 The Packing Procedure

For the packing procedure, two different versions of the algorithm were considered to solve the non-guillotine non-sequential case with rotation allowance. These were Algorithms 8 and 10 as mentioned above and as they are quite similar to each other only the latter will be explained in detail.

Algorithm 2: Main_OffURPrep

Loadorder: Order of items loaded with their coordinates

Input: List of items to choose the next item to place onto the strip

```
1   Loadorder.add(Bin)
2   Bin.OverwritePosition(X←0,Y←0)
3   Input ← Input.OrderByDescending(Largestdimension)
4   ExtremePoint First_Epoint ← new ExtremePoint(Chosen_maxdim,X←0,Y←0)
5   FitnessList ← new empty Sortedlist(Key ← double, Value ← List of Tuple of ExtremePoint and Item)
6   WHILE (|input| > 0)
7           FOR EACH( ExtremePoint E in ActiveExtremePoints)
8                   needsspace ← TRUE
9                   IF(ActiveExtremePoints>1)
10                          needsspace ← CALL Overlapcheck(E, loadorder.Last())
11                  ELSE
12                          needsspace ← FALSE
13                  ENDIF
14                  IF(!needsspace OR E.Space.Count EQUALS 0)
15                          CALL E.Create_Space(Fetchrelevantvertices(E))
16                  ENDIF
17                  currentpack ← input.First()
18                  currentpack_rotated ← currentpack.Rotate()
19                  CALL currentpack, currentpack_rotated .OverwritePosition(X←E.X, Y←E.Y)
20                  fits1 ← check if currentpack fits
```

```
21                      fits2 ← check if currentpack_rotated fits
22
23                      IF(it does not fit anywhere)
24                              CONTINUE to the next Extreme Point
25
26                      ELSE IF(it fits in any orientation)
27                              fitness1 ← calculate fitness of the item
28                              fitness2 ← calculate fitness of the item in rotated orientation
29                              IF(fitness1>fitness2)
30                                      FitnessList.Add(fitness1, currentpack, E)
31
32                              ELSE
33                                      FitnessList.Add(fitness2, currentpack_rotated, E)
34                              ENDIF
35                      ELSE IF(fits only one orientation)
36                              fitness ← calculate fitness of the item in fitting orientation
37                              FitnessList.Add(fitness, fitting item, E)
38                      ENDIF
39              END FOR EACH
40              IF(|FitnessList| EQUALS 0)
41                      RETURN
42              ENDIF
43              chosenpack ← FitnessList.First().Package
44              chosenE ← FitnessList.First().ExtremePoint
45              Loadorder.Add(chosenpack)
46              CALL Refresh_ExtremePoints(chosenpack, chosenE)
47              CALL Refresh_Vertices(chosenpack, chosenE)
48              FitnessList.ClearAll()
49              input.Remove(chosenpack)
50              IF(|loadorder| %10 EQUALS 0) CALL Updatedimensions(input) ENDIF
51      END WHILE
52      RETURN
```

The Algorithm 1 presented earlier corresponds to the line 46 of the packing algorithm for refreshing the extreme points and removing the ones not feasible anymore. The items are sorted depending on a criterion such as their largest dimension as presented here on line 3. This can be the volume, the perimeter length, the largest dimension, width, height, or a combination in case the first sorting feature is the same. The focus of the original Extreme Point heuristic from Crainic et al. 2007 was these sorting rules for the input items which was not the focus of this implementation, and a large majority of the solutions were done using only the largest dimension criteria. In the case multiple items had the same largest dimension, the order they were given in the instance file would be the sorting criteria. The reason for this was that other parameters used in the fitness function were simply more effective in the packing quality than the sorting rule, which often led to very similar sortings. The items with the largest dimensions often have also the largest volumes and the sorting rule difference only moves some items to some positions in the input sequence which can be done much more effectively in a local search phase on the sequence as all metaheuristic models do.

3.3 The Fitness Calculation

The fitness function corresponds to the merit function of the original implementation of the heuristic. The original heuristic implemented multiple merit functions to see which one performed well. These were “Minimize the free volume after accommodating an item (FV), minimize the maximum packing size on the X and Y axes (MP), level the packing on X and Y axes (LEV), maximize the utilization of the EPs’ Residual Space (RS)” Crainic et al. 2007. FV is implementing simply the volumetric 1D measurement of the bin, e.g. if the bin has 250 capacity left and we insert 60 then there is 190 capacity remaining. The MP case tries to enforce compactness on the entire packing by measuring increases in envelope size. The LEV function focuses on the level on an axis so that the increase of the new positions for placement is minimal as well as proxy enforcement of compactness is justified. Finally, RS focuses on the space (a different implementation of space is the case here but still a similar logic is the case) and how to minimize the wasted space locally in this situation.

For the scope of this thesis, a fitness function that was meant to mimic all of these mentioned functions was implemented. Upon thousands of tests with instances, the conclusion on the best function for placement was a combination of functions that were represented as parameters that are weights of some value. The fitness function implemented is by far the most complex part of packing in terms of calculation time and can be seen as a modification of the touching perimeter heuristic, the LEV function, and an additional value scheme for the strip packing problem called the penalty.

The touching perimeter value is calculated from one or more vertex of an already placed item or items with one or more vertex of the candidate item. This overlap can at most be the vertex to which the candidate item is touching and not the largest vertex the candidate has to offer as two vertices can at most coincide on the length of the shorter of two. Additionally, a vertex of the candidate, e.g. vertex 2 (right side vertical), can touch multiple vertices of multiple items at the same time and the overlap is calculated for all of the vertices that might be touching the candidate on V2. Similarly, all of the overlaps are calculated and added together to have an “overlap” value. While calculating the individual overlaps, the values are multiplied by their corresponding weight so if the preference is the left overlaps they are multiplied by a higher value, if it is the right then the right overlaps the values are multiplied by a higher value. In order to keep the options limited, the multiplier can take any integer value between 0 and 9 inclusive.

The disadvantage of this touching perimeter value is that it does not care for the remaining wasted space all that much. Especially in the cases where the candidate item to be placed finds an open ceiling meaning the space of the Extreme Point has not got any already placed items around the top to coincide with a candidate. Additionally, as the value relies on packed items to exist in the space and not on the candidate item, there is virtually no difference between a small and a large item that both perfectly cover the same existing vertices.

Simply calculating the overlap values and packing according to those is a viable option if the vertices of the bin are not considered for these overlaps. When they are considered the overlap preferences often pile the items up on the left or right side climbing similar to stairs without actually enforcing any compactness. To negate this the implementation also has a height value where the Extreme Points are given points for where they are in the strip which can be seen on line 37 of Algorithm 3. This height value is calculated as a double lower than 1 representing a percentage and is then multiplied by a parameter beta. This height value can make the difference of preferring a lower Extreme Point over a higher one given the overlap values are the same or negligibly different. Similarly, a rewarding parameter R was implemented to reward the perfect overlaps by a small double value. If the existing vertex overlaps on both ends with the vertex of the candidate item this has a higher value than just the endpoints of these vertices overlapping. The reward scheme creates the incentive to level such as in the LEV function, but also does not justify avoiding a good packing position simply because the candidate overlaps perfectly on one vertex in some other position.

The slight preference for lower positions and perfect overlaps was not enough to avoid packing items perfectly leaning on a wall such as the left vertex of the strip. For this reason, a penalty scheme was developed to penalize the increase in the strip height. This scheme only focuses on the increase of the strip height, e.g. if a position will clearly be chosen for an item but the overlap values are the same due to how the already packed items are positioned then the penalty scheme can help decide which orientation of the item is to be preferred; as can be observed in Figure 10. The penalty scheme did not make sense to have activated in the beginning but a gradual activation of it proved itself to create good results. For this gradual activation and amplification of penalties a gamma value, two alternative sigmoid curves (Figure 11), and a value assigning where the top of these curves would be, OPT, were chosen. Here the OPT value is initiated with the known best solution for the height and is offset from that for the best results where this offsetting will be explained in further detail later on.

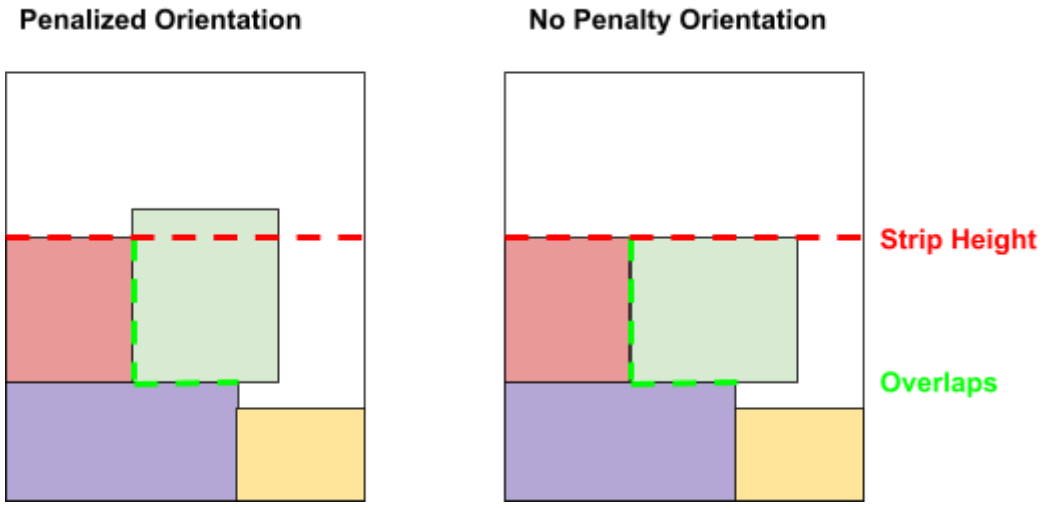


Figure 10

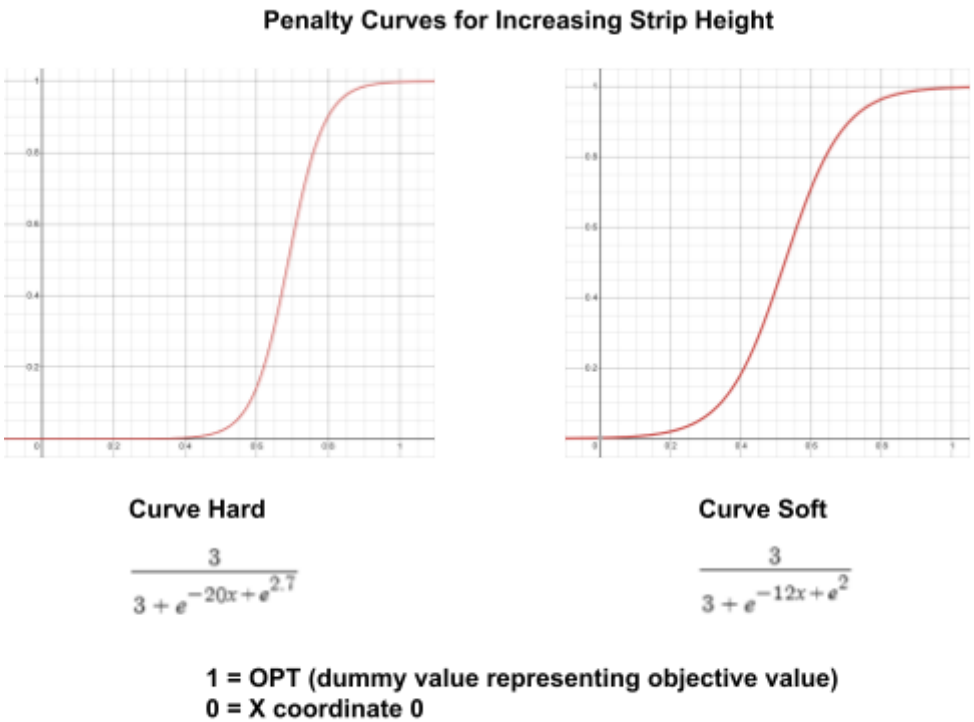


Figure 11

With the penalty scheme, in the early stages of packing the height increases to the strip are disregarded as often the larger items are packed first and this scheme would directly contradict this. Gradually as the packing proceeds the penalty becomes more significant and eventually is multiplied by 1*Gamma. With the implementation of all these parameters, different packing heuristics such as bottom left can be mimicked to an extent.

Algorithm 3: Fitness

(Package p, ExtremePoint chosen)

```
1   //SET OUTPUT VALUES//
2   output = 0; overlaps ← 0; heightvalue ← 0;  penalties ← 0; rewards ← 0;
3   //SET PARAMETER VALUES//
4   a ← new array[4]{A1, A2, A3, A4}; beta ← Beta; gamma ← Gamma; boolean rewarding ← R;
5
6   entering  ← new array of Vertex[4]{p.v1,p.v2,p.v3,p.v4}
7
8   //OVERLAPS CALCULATION//
9   FOR(i=0 to incl. |V|)
10    IF(a[i]>0)
11      overlapval ← 0
12      List<Vertex> collinearlist  ← CALL Fetchcollinear_vertices(entering[i])
13      IF(collinearlist.Count > 0)
14        FOR EACH (Vertex Ve in collinearlist)
```

```
15          overlapval ← overlapval +  a[i] *(CALL CalculateOverlap(Ve,entering[i]))
16          IF(Both ends touch) AND (rewarding))
17              rewards ← rewards + 0.5
18          ELSE IF (Ends touch) AND (rewarding))
19              rewards ← rewards + 0.2
20          ENDIF
21        END FOR EACH
22      ENDIF
23      overlaps ← overlaps + overlapval
24    ENDIF
25  END FOR
26
27  //PENALTIES CALCULATION//
28  raisepercentage  ← CALL PenaltyCurve((chosen.Y + p.Length) / Opt)
29  IF((chosen.Y + p.Length) > StripHeight)
30    penalties ← penalties + ((chosen.Y + p.Length) − StripHeight) * gamma * raisepercentage
31  ENDIF
32
33   //HEIGHTVALUE CALCULATION//
34   y1 ← ActiveExtremePoints.MaximumYCoordinate
35   y2 ← ActiveExtremePoints.MinimumYCoordinate
36
37  heightvalue ← ((y1 − chosen.Y)/(y1 − y2)) * beta
38  output ← (overlaps + rewards + heightvalue − penalties)
39  RETURN output
```

4 Parameter Tuning and Sensitivity Analysis

In this chapter the heuristic is utilized further from constructing a single good solution to being used in combination with a local search and metaheuristic scheme to fit the construction better to the fixed sequence of items input. During this process many solutions are constructed and their quality is evaluated. The sequence is maintained but the algorithm tries to pack these items better with each iteration. There is a temperature parameter that also enters the local search operator to limit how much exploration that is allowed when the objective value is the same or worse. This scheme was used to identify patterns for good packing without having the parameters inclined to converge towards specific values like it could be done with a regression approach. The simulated annealing approach is kept quite standard but the local search operator had to be custom built as to decide when to move which parameter and to stop according to the objective.

It is quite the standard in most papers to simply state which parameters were chosen for the algorithm for which reasons and move on to computational results. In the case of this construction heuristic, with so many parameter combinations to choose from, simply choosing fixed parameters would be limiting the algorithm's performance. Construction heuristics build upon simple ideas of fitness such as the cheapest insertion or farthest/nearest neighbor for the traveling salesman

problem (TSP). As soon as other constraints are included the simple construction becomes not as simple to implement as they also need to be considered at each step.

In the case of this heuristic, these ideas are depicted as a set of parameters. Similarly, Burke et al. 2004 have the idea of inserting new items according to existing neighbors of the space such as the tallest neighbor, shortest neighbor, and smallest neighbor. This can be mimicked only to some extent as the EPSP is a bottom left placement-based algorithm only allowing the lower left corner of an item to be placed on a point with only consideration to the upper right space around this point. The preference values of overlap A1=bottom, A2=right side, A3=top side, A4=left side (Algorithm 3 line 4) can be used to manipulate the positioning of items but having a high A2 alone does not mean that the item will be placed touching the right neighbor if there exists no Extreme Point allowing this positioning. Additionally, the reward, the position height, and the penalty schemes work hand in hand with the preference values, very similar to a constraint affecting the choice of the nodes in a vehicle routing problem (e.g. time windows), here these schemes represent the interest of having a low strip height. The bin/strip is also represented as an item so that the overlaps to the bin also are calculated in the fitness. This causes the items to pile up on one side of the bin/strip if there is no limitation on height, an incentive to have perfect overlaps or choose lower positions.

The unique problems that present themselves here are that firstly the values of overlaps in the array A differ for good results on different instances, secondly, there is no observable linear relationship between these values so that a regression approach would provide consistently good results, and lastly changing values may not change the objective value, especially in larger instances.

Another detail is that simply moving a value does not indicate that the quality will increase or decrease continuously, e.g. moving A2 from 4 to 5 might improve objective value while moving it from 5 to 6 might worsen it and 6 to 7 might improve it even further. This is due to the multi-dimensional nature of packing problems where the sequence of items is not the only important aspect but where and how these items are packed can affect the rest of the packing drastically causing a domino effect of alternative packing positions for parts or the entire rest of the sequence. Each move of a value and combination of values may create unique solutions. Which parameter value is chosen first to make a change on may also land in a neighborhood of values completely different. This makes the issue of parameter tuning significantly harder and needing a custom approach which will be discussed in detail.

Initially, the Gamma value (used in Algorithm 3 line 30), and the Beta value (used in Algorithm 3 line 37) were chosen initially to be 30 arbitrarily (after some observations), and the OPT value (used in Algorithm 3 line 28) was chosen to be the best-known solution or the optimal value. The values of A were limited to integer numbers within the range of 0-10 to limit the options that may be explored. While good results could be achieved simply by choosing the A values, for consistently good results it was clear that some patterns needed to be identified in the A values.

4.1 Simulated Annealing for Parameters

Due to the above-mentioned details on the parameters, the tuning task was not as simple as traditional approaches to parameter tuning. In order to find good fitting parameters, some of them were fixed and some of them were allowed to be changed randomly in random directions. This was all done within a simulated annealing algorithm where the local search element in the algorithm also had the temperature as an input to decide on moves on the parameters.

The algorithm presented in Algorithm 4 starts exploring the possible combinations of parameters offsetting from the input parameters until an iteration or time limit is reached. In the cases where the quality is the same or better, the new parameters are adopted (Algorithm 4 lines 18-25). If the quality is worse a temperature and quality difference-based decision is made (Algorithm 4 line 28). Depending on the number of explorations done so far either a previously explored set of parameters is chosen for the next iteration or a totally random set of parameters is created for the next iteration (Algorithm 4 lines 32-42).

Algorithm 4: Simulated Annealing

(TimeLimit)

```
1   timer.Start()
2   Bestval ← int.MaxValue
3   iteration ← 0
4   limit ← 2
5   Temperature ← 5
6   InitialTemp ← Temperature
7   alpha ← 0.95
8   Parameter[] par_internal ← initialparameters
9   incumbentobjval ← Bestval
10  WHILE (iteration < limit AND timer < TimeLimit)
11    par_internalcopy ← par_internal.Clone()
12    incumbentobjvalcopy ← incumbentobjval
13    CALL Parameter Local Search(ref par_internalcopy, ref incumbentobjvalcopy, Temperature, TimeLimit)
14    Key_value ← create a key value from par_internalcopy
15    IF (Neighborhood_sofar does not contain this Key_value)
16      CALL Add_globalhistory(par_internalcopy, incumbentobjvalcopy)
17    ENDIF
18    IF (incumbentobjvalcopy is same or better than Bestval)
19      Bestval ← incumbentobjvalcopy
20      incumbentobjval ← incumbentobjvalcopy
21      par_internal ← par_internalcopy
22    ELSE IF (incumbentobjvalcopy is same or better than previous incumbentobjval)
23      incumbentobjval ← incumbentobjvalcopy
24      par_internal ← par_internalcopy
25    ELSE //the objective value worsened
26      decision ← random.NextDouble()
        diff ← incumbentobjvalcopy - incumbentobjval
```

```
27      currentstate ← e^(−diff÷Temparature)
28      IF (decision <= currentstate) //acceptance
29        incumbentobjval ← incumbentobjvalcopy
30        par_internal ← par_internalcopy
31      ELSE //refusal
32        IF (Neighborhood_sofar has sufficient data in it) //will choose a random neighbor from existing neighborhood
33          index ← choose random index in the Neighborhood_sofar
34          par_internal ← Neighborhood_sofar[index].Key
35          incumbentobjval ← Neighborhood_sofar[index].Value
36        ELSE //will choose a random neighbor with random values
37          incumbentobjval ← int.MaxValue
38          par_internal ← create random parameter values
39        ENDIF
40      ENDIF
41    ENDIF
42    Iteration ← iteration +1
43    Temperature ← Temparature*alpha
44  END WHILE
45  timer.Stop()
46  RETURN
47
```

The local search operator moves one of the parameters randomly and runs the construction algorithm to see its effect. Depending on the output quality the next moves are decided. If the quality is better the algorithm continues moving the parameter in the same direction (+ or -) until either a max iteration limit is reached for that specific parameter, the solution quality is not improving anymore, or a max iteration limit for repetition of the same objective is reached.

When the algorithm repeats the same objective value without improving for a long time (5 iterations) the entire search is broken (Algorithm 5 line 49). Otherwise, the decision is made to change the direction (if the parameter value was increasing it starts decreasing) and then to move on to another parameter (Algorithm 5 lines 52-58). The stepsize of moving (e.g. from 4 to 5 with stepsize=1 or from 4 to 6 with stepsize=2) is increased if the new objective value is worse than the old one and the temperature-based decision is to change the step size and accelerate the moves on this parameter. If the decision is to not change step size then the direction is changed given that it was previously not changed on the parameter (Algorithm 5 lines 70-77). If the steps were large and the solution was refused then the stepsize is set back to 1 and the direction is reversed(Algorithm 5 lines 78-82). At the end of these moves (and an iteration limit) the parameter is locked out for the rest of the local search and only the remaining parameters are considered for moves (Algorithm 5 line 11). Figure 12 demonstrates how the parameters are chosen and moved where the red indicates the current parameter choice and the dotted box is the values that specific parameters may take. The parameter OPT was tested with 3 and 6 percent and depending on the instance size this percentage was changed from the initial value. The choice of the parameters also considers different probabilities as some parameters are more relevant for the solution quality

such as A2 and A4 for right and left overlaps. The decision conditions in algorithm 5 are highlighted green for better solution, yellow for same quality of solution, and red for worse quality solution for better readability

Algorithm 5: Parameter Local Search

(parameters, currentval, Temperature, TimeLimit)

```
1   par_copy ← parameters.Clone()
2   internal_Bestval ← currentval
3   Neighborhood_LS ← new Dictionary<int, List<int[]>>() // obj and parkey
4   parkeyinit ← create a key value from parameters
5   Parameters_repeating_objectiveval ← new List<Parameter[]>()
6   repeated objective value← 0
7   Bestarray ← par_copy
8   boolean unbrokenloop ← true
9   WHILE (unbrokenloop AND timer < TimeLimit)
10    choice ← random.NextDouble()
11    index ← CALL Choose(par_copy, choice); //chooses which parameter to move or break out of the while with “unbrokenloop”
12    iteration ← 0
13    direction ← random.Next(0, 2)
14    previously changed direction  ← 0
15    stepsize ← 1
16    WHILE (iteration < maxiteration) //each parameter has a max iteration as some parameters have smaller ranges for movement
17      oldobj ← currentval
18      par_copy[index].UpdateVal(direction, stepsize) //move the chosen parameter in the current direction with current step size
19      parkey ← create a key value from par_copy
20      newobj ← int.MaxValue
21      IF (Neighborhood_sofar does not contain this parkey)
22        Solver.Timelimit ← TimeLimit - timer.ElapsedMilliseconds //we give a fixed time to the construction to manage the packing
23        newobj ← CALL Solver.RunNewPars(par_copy)
24        Neighborhood_sofar.Add(parkey, newobj)
25      ELSE
26        iteration ← iteration +1
27        CONTINUE
28      ENDIF
29      IF (Neighborhood_LS does not have newobj)
30        Neighborhood_LS.Add(newobj, parkey)
31      ELSE
32        Neighborhood_LS[newobj].Add(parkey)
33      ENDIF
34      IF (newobj is better than oldobj)//if better
35        Parameters_repeating_objectiveval.Clear()
36        repeated objective value← 0
37        IF (newobj < internal_Bestval)
38          internal_Bestval ← newobj
39          Bestarray ← par_copy
40          IF (newobj <= Bestval) //if we find a value better than the best value we get out of LS
41            Bestval ← newobj
```

```
            Add_globalhistory(par_copy, Bestval)
            iteration ← maxiteration
            BREAK
          ENDIF
      ELSE IF (newobj EQUALS oldobj) //if same quality
        repeated objective value ← oldobj
        Parameters_repeating_objectiveval.Add(par_copy)
        IF (|Parameters_repeating_objectiveval| > 5) unbrokenloop ← false; iteration ← maxiteration; BREAK; ENDIF //get out to stop
repeating
        whattodo ← (Temperature / InitialTemp)
        IF (whattodo < 0.60)  //refuse these parameters
          move in case of refusing  ← random.Next(0, 2)  // 0=dont change direction, 1=change direction
          IF (move in case of refusing  EQUALS 1)
            Undo the last movement of par_copy[index]
            IF (previously changed direction  EQUALS 0)
              direction ← the other direction //either increase or decrease
              previously changed direction  ← previously changed direction  +1 //save that we changed direction on this parameter
            ELSE
              iteration ← maxiteration
            ENDIF
        ELSE IF (stepsize > 1)
          stepsize ← stepsize -1
        ENDIF
      ELSE IF (newobj is worse than oldobj) //if worse quality
        diff ← newobj - oldobj
        whattodo ← (oldobj − diff) ÷ oldobj * (Temparature ÷ InitialTemp)
        IF (whattodo < 0.75) //refusing the worse objective
          Undo the last movement of par_copy[index]
          IF (stepsize is less than the allowed step size for this parameter)
            move in case of refusing  ← random.Next(0, 2)  // 0=dont change direction, 1=change direction
            IF (previously changed direction  > 0)
              stepsize ← stepsize +1   //cant change direction anymore so we go faster in the direction we are using currently
            ELSE IF ((move in case of refusing  EQUALS 1) AND direction was not changed so far on this parameter)
              direction ← the other direction
              previously changed direction  ← previously changed direction  +1
            ENDIF
          ELSE //stepsize limit is reached for this parameter, already going faster than 1 step
            decrease step size
            IF (direction was not changed so far on this parameter)
              direction ← the other direction
              previously changed direction  ← previously changed direction  +1
            ELSE
              iteration ← maxiteration
            ENDIF
          ENDIF
        ELSE  stepsize ← 1 //not refusing the worse objective
      ENDIF
      iteration ← iteration +1
    END WHILE
For this parameter the Local Search is completed ← true
END WHILE
```

93	parameter_output ← randomly one of the parameter sets from the best objective value key in the Neighborhood_LS
94	as reference output: currentval ← Neighborhood_LS.Best.ObjectiveValue
95	as reference output: parameters ← parameter_output
96	**RETURN**

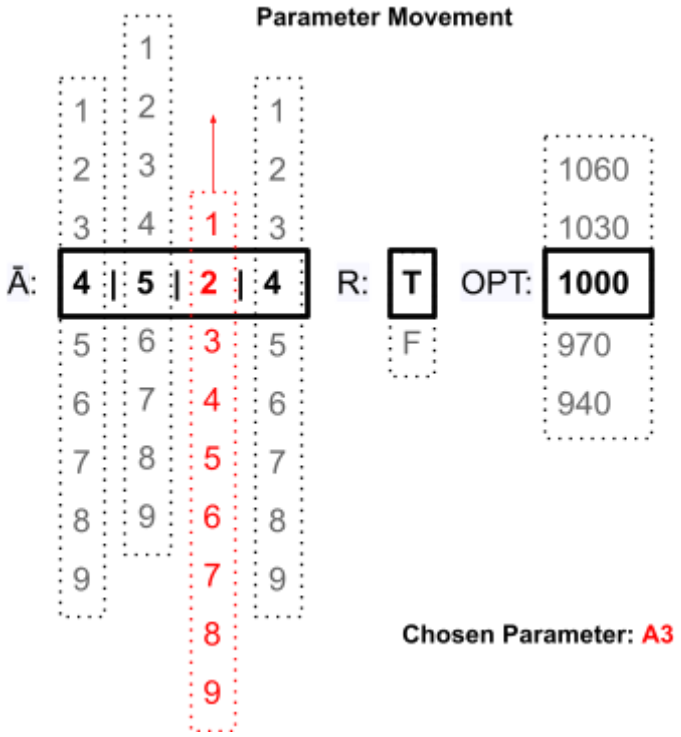


Figure 12

This simulated annealing algorithm with the local search on parameters proved to create very good results as will be mentioned in detail later on. The algorithm can create very good results within the first few iterations, and even when the starting parameters are far from optimal (e.g. A = [0,0,0,0]) the algorithm can reach the desired results within a few iterations. The complication related to this scheme is that larger instances take a long time due to the number of positions possible and this local search operation at each iteration runs the construction algorithm multiple times if the combination of parameters is untested (Algorithm 5 line 23). The advantage is that one can test some parameter patterns and their close neighbors with ease when the algorithm runs parallel on multiple cores. The disadvantage is that this approach takes a long time on larger instances due to the nature of the construction algorithm regardless of how efficiently the multiple cores and memory are utilized.

4.2 Sensitivity Analysis of Construction Parameters

The computation of fitness for specific positions in the strip at any given step of the algorithm is the most time-consuming part and can be clearly observed to be a bottleneck under specific conditions. The fitness function presented in Algorithm 3 uses all of the parameters mentioned in the chapters above to calculate the best position for the next item. The higher the number of

calculations, the longer this function takes to run and as the algorithm does not run with a first fit approach (choosing the first position where the item fits, possibly sorting the Extreme Points by height value prior to positioning the item) but rather with a best-fit approach (choosing the best Extreme Point to position, with no effect of prior sorting other than the cases where the fitness score would be the same. Has no relation to the BF heuristic). The calculation time of each fitness position and sorting of these values increases or decreases with modifications to the parameter values for the fitness value.

An Analysis of the sensitivity to the parameter values was done using Path and Nice instances from Dr. Christine Mumford (also present in the literature under Valenzuela) which were created for Metaheuristics International Conference in Porto (MIC 2001). These instances have each 6 files with originally float numbers as they were obtained by noninteger cuts but were then rounded by researchers who are unknown to Dr. Mumford. The instances with rounded values are referred to as the originals and have been published with altered lower bounds which can be found in the computational results of this thesis. The Path instances have very few items with very irregular shapes that can be very long and thin or have a large variance of volumes while the Nice instances have items with similar dimensions and volumes. The instances begin with 25 items and go up to 1000 items per instance. These instances were chosen as they are sister instances and give a good insight into what effect a parameter has that can be clearly observed both visually and in terms of computational time.

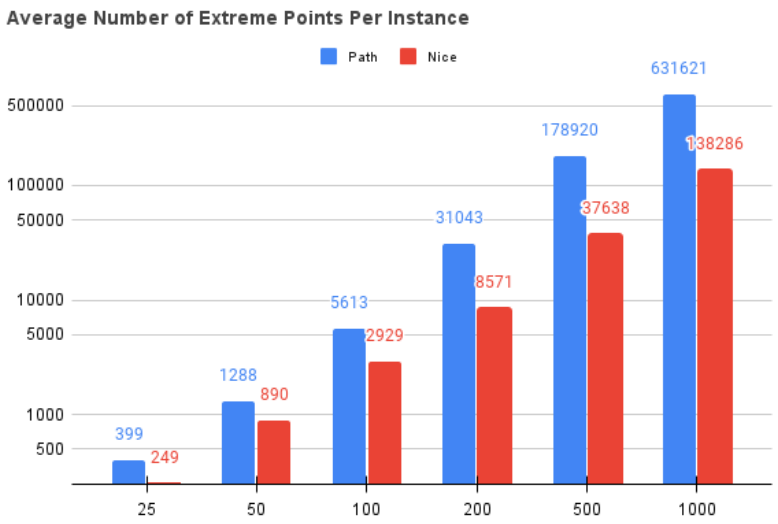


Figure 13

With the use of the simulated annealing algorithm presented in the section above the instances were tested to see how they scale. For this, the rewarding scheme was not forced but there was a probability of 10% that the algorithm would change whether it would initiate it or not. The observation here was that the rewarding scheme would be turned on and off frequently enough for the results to be representative of how the instances scale when tested for various parameter values. The results presented in Figure 13 demonstrate how the irregular shapes of the Path

instances lead to a drastic increase in the number of possible positions for placement as they create more shadow points and more indents in the packing pattern. It is also clear from these results that the extreme point heuristic gets bottlenecked by its own fill feature as it has to consider the entire strip at all times.

The reward scheme was chosen in the end to be a parameter that can be manipulated. The reason for this was the insignificant change in the number of Extreme Points other than the cases where the instances are larger and there is a strong variance in item volumes and dimensions. When looking at the Path and Nice instances, as presented in Figure 14, it becomes clear that the reward scheme improves the speed on large instances like Path5 (n=500)and Path6 (n=1000). Conversely, this effect cannot be clearly observed on the larger instances of Nice, and while the results may also be affected by the starting parameters of the algorithm and the runtime constrained on the simulated annealing to find these averages the conclusion here is that forcing this scheme to be turned on or off does not benefit the objective. The guillotine cut property of these instances also may be contributing to these results and distorting the performance of this scheme and others shown in this chapter.

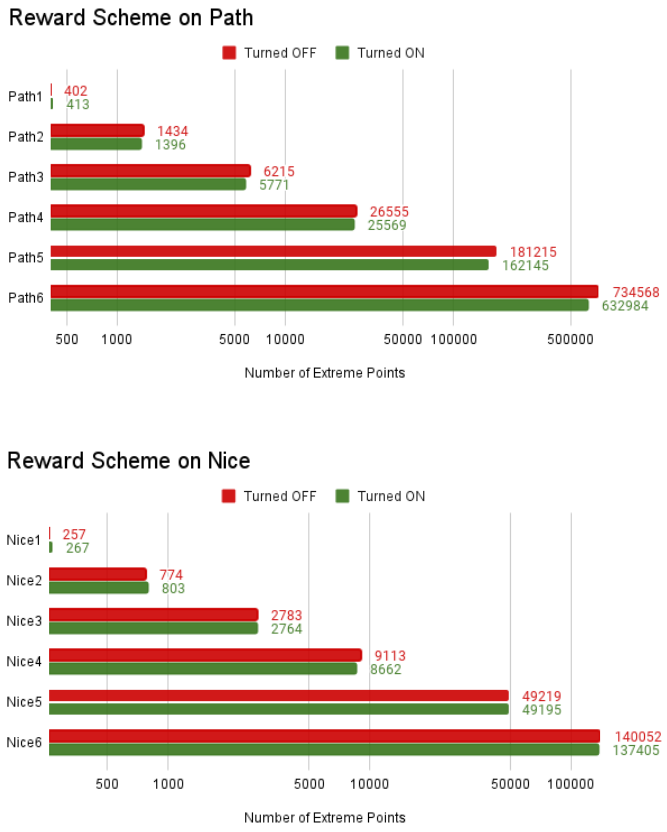
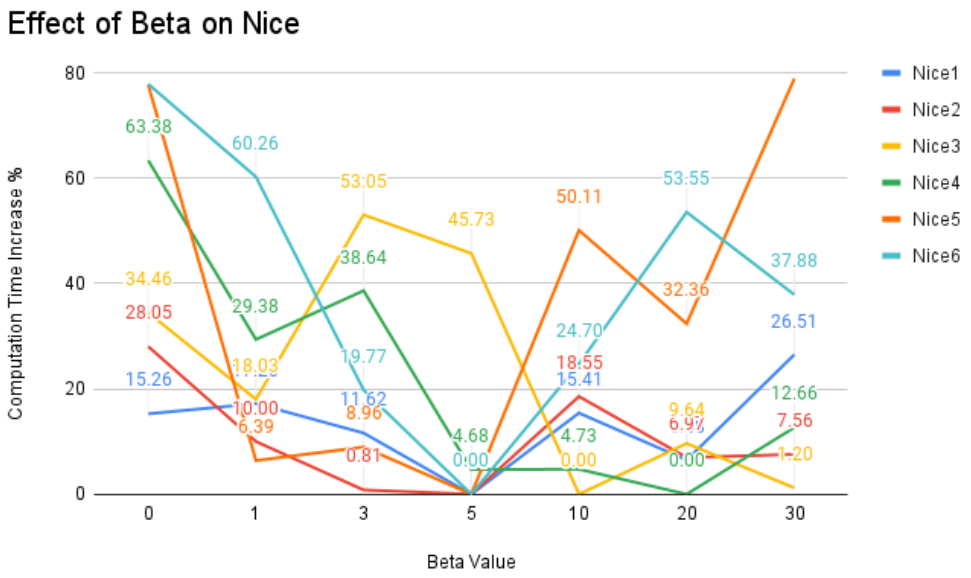
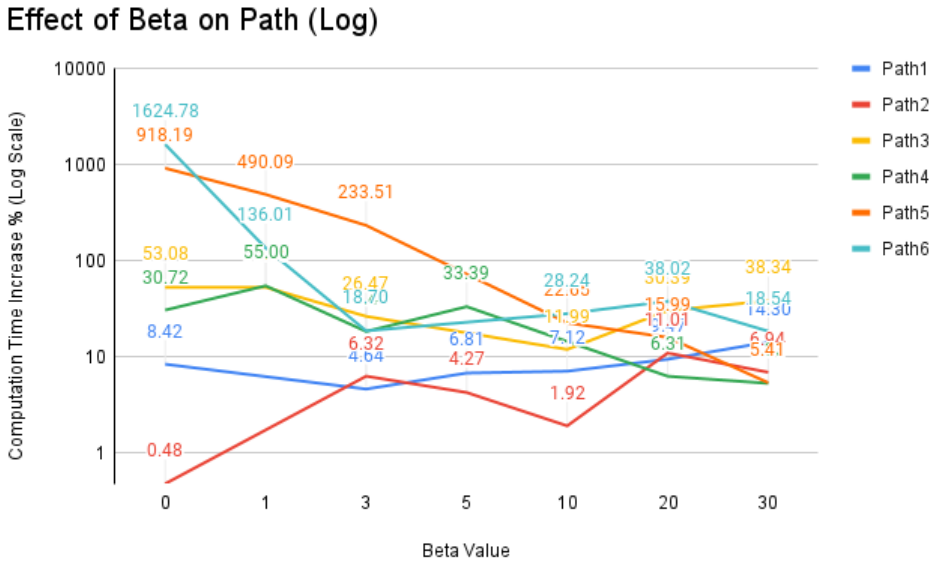


Figure 14

The Beta value that affects and amplifies the importance of the Y value of an Extreme Point has a strong effect on the computation time of the fitness score and consequently the total packing time. The gamma value was fixed at 30, the penalty curve was set as the hard sigmoid curve, and the sorting was done by the largest dimension for all the instances in Path and Nice. Figure 15

demonstrates the effect of changing the beta value and in the first chart, it can be clearly seen that the beta value 5 has an advantage over other values for the Nice instances. More than 75% of all instances while testing had their best objective value when the beta was set between 1 and 5 while half of all instances had their best objective value at beta set as 5. It can clearly be observed that the effect is much stronger on the Path and on Path6 the computational time increases as high as 1625 percent. The findings from the analysis are that the beta value should be set at a positive value, preferably between 1-10 and the height value scheme not only contributes to the objective value but also makes the algorithm run faster, increasing the chances of finding the optimal value if used in combination with a local search operator in a metaheuristic algorithm. The second chart in Figure 15 does not show the lowest values (0 percent) as the vertical scale is logarithmic and the value of log(0) is not defined.





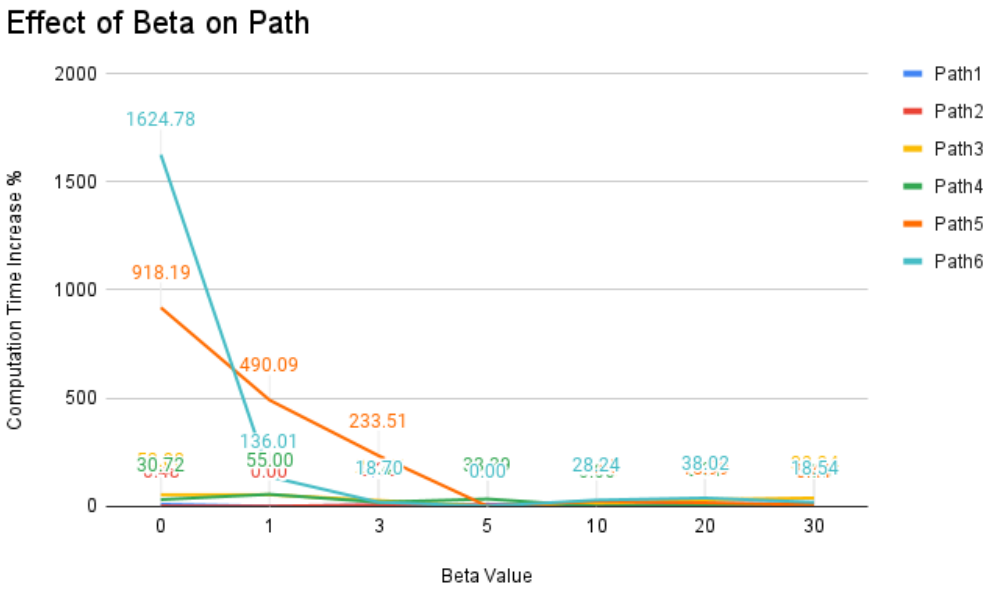


Figure 15

The Gamma value for the penalty was also analyzed using the beta value 5 and the item sorting by largest dimension. The data from the analysis was sorted in the charts in Figure 16 to group the instances with 25, 50, and 100 items and the instances with 200, 500, and 1000 items. The top charts present the effect of the gamma value when using a hard penalty sigmoid curve and the bottom charts present the effect when using the soft curve. The findings from this analysis was that the value 5 and 30 were good in quality and speed with the latter being good with larger instances and the former with smaller instances. The soft curve was not utilized in the testing of instances outside of the Path and Nice instances but when using the softer curve similar quality in terms of objective value and time could be observed in the range of 5-30. In the end, the Gamma value 30 was generally adopted for further testing as the objective value quality improvement was justifiable considering the computational time tradeoff for some instances.

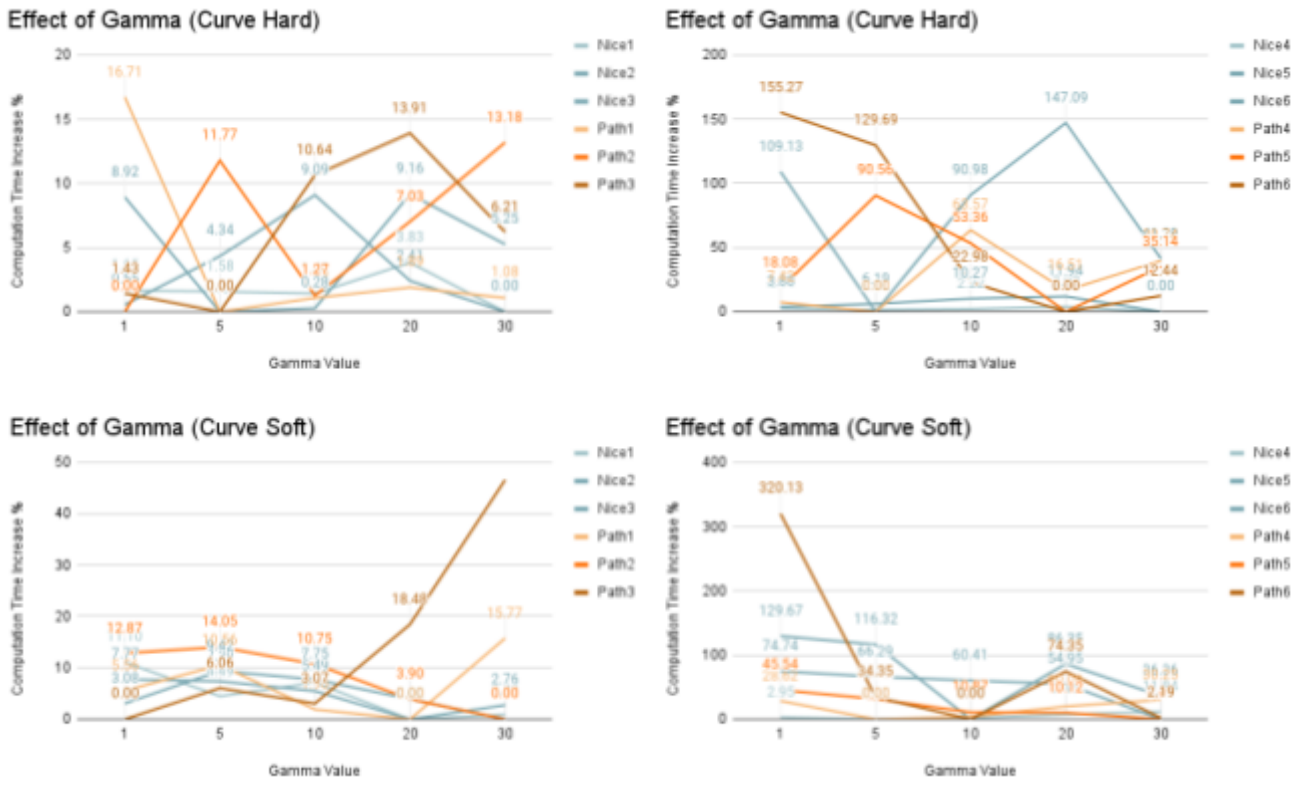


Figure 16

5 Computational Results

The EPSP construction algorithm for packing and the simulated annealing algorithm were tested on all of the instances in Table 1. Some of these instances are zero waste and some of them are non-zero waste. This means that the items are either acquired by cutting a predecided strip rectangle into smaller rectangles and creating, therefore, a wasteless strip when packed optimally, or they allow for wasted space when packed optimally as they are not a perfect guillotine cut result of a larger rectangle. The instances that have wasted space do not strictly have an optimal value provided but the height is provided as a lower bound. The Path and Nice instances used intensively in this thesis have modified optimal values due to the rounded item dimensions as mentioned earlier. These values can be seen in detail in the results tables or can be found in Yang et al. 2013.

Authors	Instance Set Name	Number of Instances	Number of Items	Width	Height	Wasted Space
Hopper & Turton 2001	C	21	16-197	20-160	15-240	☐
Hopper E. 2000	N	35	17-199	200	200	☐
Hopper E. 2000	T	35	17-199	200	200	☐

Authors	Instance Set Name	Number of Instances	Number of Items	Width	Height	Wasted Space
Burke et al. 2004.	Burke	13	10-3152	30-640	40-960	☐
Wang & Valenzela 2001	Nice	6	25-1000	1000	1000-1001	☐
Wang & Valenzela 2001	Path	6	25-1000	1000	1000-1002	☐
Beasley 1985a	ngcut	12	7-22	10-30	23-87	☑
Beasley 1985b	gcut	13	10-50	250-3000	1016-12522	☑
Christofides & Whitlock 1977	cgcut	3	16-60	10-70	23-636	☑
Bengtsson 1982	beng	10	20-200	25-40	30-156	☑
Ramesh Babu & Ramesh Babu 1999	babu	1	50	1000	375	☑

Table 2 : Instances Used for Testing

The packing was compared against other construction algorithms first with the Burke, babu, and C instance sets. At this stage, both algorithm 8 Off-|UR| and algorithm 10 Off-|UR|-Prep were used with parameters that were arbitrarily but intuitively chosen (strong and weak left, right or bottom preference). As a reminder, algorithm 8 calculates all possible positions for all possible items at each iteration, and algorithm 10 calculates the fitness of the next item online for all possible extreme points at each iteration. The “Off” in the name simply suggests that the input order of items may be modified and the modification of algorithm 10 is the “Prep” referring to the reordering of items according either to their volume or to their largest dimension prior to beginning the packing. Algorithm 8 was soon thereafter abandoned for further testing as it was clear that beyond a certain number of items it was scaling poorly, especially if the algorithm is ever to be used in combination with a metaheuristic as this poor scaling would bottleneck such an approach immensely.

In the tables, algorithm 8 is referred to as EPSP|8|, algorithm 10 as EPSP|10|, and the combination of algorithm 10 with simulated annealing as EPSP-SA, all of them highlighted in yellow. The simulated annealing was used with different iteration limits due to time constraints and as a reference, the limits on Path and Nice can be observed in the repository solution files. The simulated annealing ran parallel on 5 cores on a laptop equipped with AMD Ryzen 4600H (base 3 GHz, boos 4 GHz) with 16GB DDR4 RAM, yet the algorithm never rose above the use of 3GB. The comparisons done with the other metaheuristics are limited to 10 runs of 60 seconds each while for bigger instances than 200 items this had to be exceeded (see N13 computational times). The cores had the same parameter values as a starting point but as the algorithm employs randomness for the movements of parameters this was not a serious concern.

Name	n	Optimal	EPSP|8|	Time(s)	EPSP-SA	EPSP|10|	Time(s)	BF	BBF	BL-DH	BLF-DH
RB	50	375	400	0.668		400	0.108	400	400	-	-
C7.3	196	240	244	7.316	242	245	1.52	245	243	257	249
C7.2	197	240	243	7.588	241	243	1.572	244	242	264	247
C7.1	196	240	244	26.364	243	243	0.911	247	243	252	249
C6.3	97	120	122	1.283	122	123	0.254	124	123	131	128
C6.2	97	120	123	1.495	121	122	0.212	122	123	130	123
C6.1	97	120	123	3.245	122	123	0.236	123	123	130	126
C5.3	73	90	92	0.379	91	93	0.155	93	92	97	94
C5.2	73	90	91	0.273	**90**	92	0.129	92	92	99	95
C5.1	73	90	92	0.344	91	93	0.155	93	91	94	94
C4.3	49	60	62	0.509	61	62	0.117	62	62	64	63
C4.2	49	60	63	0.311	61	61	0.146	62	63	68	63
C4.1	49	60	62	0.915	61	63	0.12	63	62	67	66
C3.3	28	30	33	0.1	31	32	0.093	33	33	34	34
C3.2	29	30	32	0.144	31	32	0.086	34	33	33	32
C3.1	28	30	31	0.154	31	32	0.098	32	32	33	33
C2.3	25	15	16	0.146	**15**	16	0.04	16	16	17	17
C2.2	25	15	16	0.107	16	16	0.093	16	17	26	26
C2.1	25	15	16	0.187	16	16	0.094	16	16	17	17
C1.3	16	20	21	0.079	21	21	0.083	24	21	21	21
C1.2	17	20	21	0.053	21	21	0.085	22	21	22	22
C1.1	16	20	21	0.092	**20**	21	0.076	21	21	23	22

Table 3: C and Babu

Name	n	Optimal	EPSP|8|	Time(s)	EPSP-SA	EPSP|10|	Time(s)	BF	BBF
N13	3152	960	-	-	961	964	1412.074	964	964
N12	500	300	304	53.351	304	305	7.362	306	302
N11	300	150	151	14.042	151	152	1.913	152	151
N10	200	150	151	7.163	151	152	0.149	152	151
N09	100	150	152	11.36	153	153	0.168	152	151
N08	80	80	83	0.454	82	83	0.175	84	82
N07	70	100	103	0.509	102	104	0.118	107	106
N06	60	100	102	0.221	101	102	0.112	103	102
N05	50	100	105	0.194	104	105	0.075	105	103
N04	40	80	83	0.137	82	83	0.094	83	82
N03	30	50	52	0.112	52	52	0.096	52	52
N02	20	50	53	0.057	52	54	0.048	53	52
N01	10	40	40	0.032	-	**40**	0.036	45	40

Table 4: Burke

In comparison with Bottom Left and Bottom Left Fill (BL and BLF in table 1) the EPSP algorithms perform significantly better on the C dataset often coming one or two points close to the optimum value. Best Fit and Bidirectional Best Fit (BF and BBF in the tables) perform very well on almost all instances, with BBF performing incrementally better than BF. The modified Bidirectional Best Fit presented in Özcan et al. 2009 and 2013 aside from the non-modified version was omitted as it combines multiple items and creates larger items prior to packing. While this approach might work well for guillotine cut instances where the items are created together and may align perfectly as some must share the same dimensions it is unlikely that in reality, the stock cutting problem will deliver these guillotine cut properties.

While the arbitrarily used parameters delivered adequate results that can be seen in Tables 2 and 3 the EPSP-SA found better parameters for almost all instances, in some cases finding the optimal packing solutions. The algorithm always found a really good solution within the first 5 iterations of the simulated annealing and in the case of N13 with 3152 items the algorithm only ran a maximum of three iterations to find the value 961. Interestingly, the larger the instances get the lower the effect of the parameter values were, given the order of items created a good enough solution. This was especially the case for N13 where the worst found value was 982 and more than 60% of the tried parameter values were creating the solution 964. This observation indicates the need for multiple local search operators in the case of a sequence optimization so that the algorithm has the flexibility to decide what parts of the sequence have a higher impact when modified as opposed to randomly moving in the sequence. Given the very long runtime of N13, it is improbable that a sequence optimization would be able to reach a better solution if it acts fully random or even tries to learn from previous iterations as the number of iterations completed would be quite limited.

Name	n	Optimal	EPSP-SA	Time(s)	BF	BBF
Path1	25	1001	**1077**	0.032	1101	1091
Path2	50	1000	**1025**	0.13	1138	1074
Path3	100	1000	**1028**	0.27	1073	1073
Path4	200	1000	**1028**	2.867	1041	1053
Path5	500	1002	**1029**	25.234	1037	1032
Path6	1000	1002	1031	148.402	1028	1028
Nice1	25	1000	**1058**	0.103	1074	1083
Nice2	50	1001	**1066**	0.124	1085	1079
Nice3	100	1001	1070	0.238	1070	1067
Nice4	200	1001	1055	0.766	1053	1053
Nice5	500	1000	1034	6.351	1035	1033
Nice6	1000	1000	**1028**	32.717	1037	1037

Table 5: Path and Nice

On the Path and Nice instances, the construction algorithm can provide results better than the BF and BBF on the majority of instances. The two other construction methods have the advantage of also being able to place items with their lower right corner (P2) and therefore can create better solutions for some instances. The EPSP can still come very close to these results or create better solutions given the right parameter values.

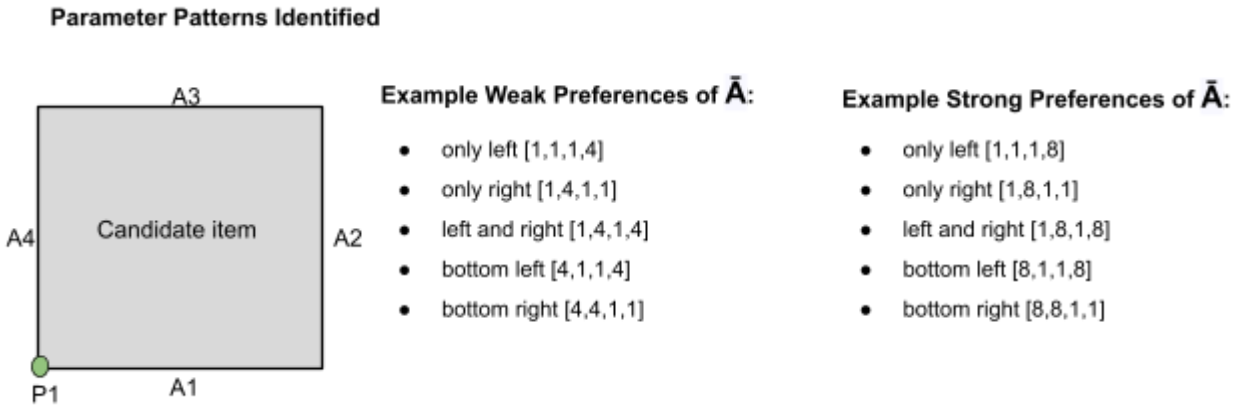


Figure 17

When it came to the right parameters found in the simulated annealing, the following patterns shown in Figure 17 could be identified. The touching perimeter value for A3 was mostly irrelevant but visually it did make a difference and improved solutions often enough to not eliminate it. The important parameters were usually the left, right, and bottom parameters and here either there would be a strong preference or a weak preference over the other A values. For example, the 961 objective value (optimal 960) for N13 was found with Ā =[2,5,1,4] where there is a weak left and right preference and the right side value is slightly higher. The algorithm generally had a higher chance of finding really good parameters when these preferences were set as a starting point e.g. Ā =[4,4,2,4] was used often for this purpose. A multi-start strategy could be promising if the different preferences could run a parallel search on the cores. This was the algorithm that manages to fit the given sequence of items into the strip in the best possible way, regardless of how the sequence is altered.

When compared with other metaheuristic methods the construction algorithm works very well but not better than methods that can modify the sequence of items. The total averages are strongly influenced by the smaller instances where the sequence has a bigger impact on the objective value as opposed to larger instances where the sequence can be obsolete in parts due to the placement policy/preferences. Figure 18 presents the results reached by EPSP-SA and the values are the best value of the 5 cores in a single run on the N and T instances. The values from a single core have to be integers that are either full or half percentages from the optimal value. Figure 19 presents the results on the non-zero waste instances and the results are the average of all cores outputs to emulate running the algorithm multiple times (as other metaheuristic results are averages). On average, for non-zero-waste instances, the ESPS-SA outperforms all metaheuristic

methods in the literature. All the results can be seen as tables with the actual values in the appendix. An interesting finding about the results of EPSP-SA and other metaheuristics is that smaller instances tend to be more challenging to find near-optimal results for.

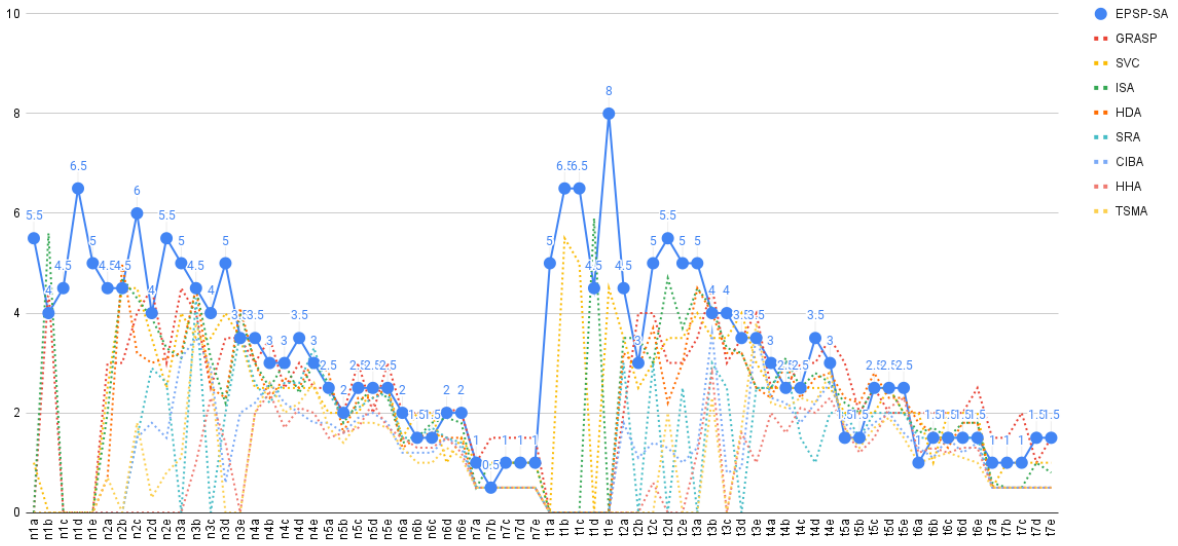


Figure 18 N and T

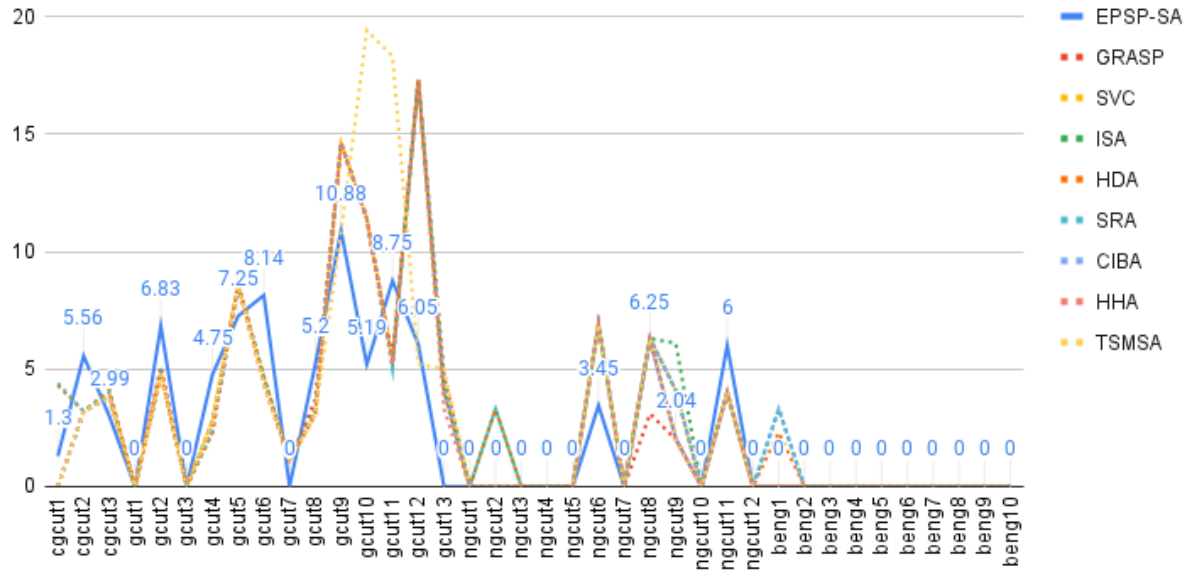


Figure 19 Non-Zero Waste Instances

6 Conclusion

The Strip Packing Problem has the unique characteristics where the solution quality is significantly affected by packing decisions unrelated to the sequence of items. Similar characteristics are present in other problems like capacitated vehicle routing problems with time windows, backhauls, consistency, multi depot, etc. but the infinite height and movements within the strip give the SPP a

much stronger independency from the sequence. This thesis aimed to prove this detachment and that there may exist opportunities to create solutions of high quality with a fixed sequence. The solution methods used here can be implemented without much challenge on real life scenarios where other systems have to output a fixed sequence which then is the bottleneck of the packing problem. The item dimensions can also be increased to three or more (geometric dimensions and even other loading constraints such as load bearing, stacking, orientation) yet even the orientation increase of 3x(from 2 to 6) should be a significant concern for the computational times and efficiency. It should be noted this thesis was only limited to the two dimensions but from the quality of solutions similar approaches could be tested with more dimensions.

6.1 Experimental Findings

The construction heuristic implemented in this thesis succeeded in creating solutions of excellent quality with fitted parameters and comparably good quality with intuitively chosen parameters to other construction heuristics which are the basis for the best known solution methods for this problem. The fitting of the parameters done with the simulated annealing and parameter local search scheme created results of very high quality, often comparable to the other state of the art methods within 60 seconds for instances under 200 items. While instances above 1000 items seemed to be much less sensitive to the parameters, the instances between 200-1000 items were sensitive enough to consider employing such a parameter local search method. Unfortunately, the heuristic implemented in this thesis requires too much time for instances above circa 300 items for it to be able to find really good solutions with high confidence within a minute. This finding is prominent especially in the case of instances with 1000 items or larger where the runtime is high enough to not consider having the simulated annealing run on them.

This thesis attains the following findings with these experimental results on literature instances: The Extreme Point Heuristic of Crainic et al. 2007 can be used for the 2DSPP and can create good results, matching the quality of other construction heuristics for this problem. The sequence is highly important for the SPP but it is not the only important factor for good packing solutions. There exist other methods to pack items efficiently that are disconnected from the sequence. The idea of spaces from Krebs et al. 2022 can be used to increase speed in the algorithm but the implemented version of iteratively exploring the vertices as rules seems to be slowing down the advantages brought by this idea.

The Extreme Point approach is significantly bottlenecked by the “Fill” characteristic of it as it considers below the skyline for packing. This scales very poorly and is the main reason for the stark increase in computational times for the larger instances. While this increase cannot be controlled without modifying this characteristic, the implementation can be done more efficiently to bring down the times required for larger instances under 1000 items. This could be done with modified fitness functions, disregarding of shadow and float points, use of approximation or

vectorization. These approaches were not possible in the scope of this thesis due to time constraints.

6.2 Further Research

The challenges encountered during the development of the algorithms explained in earlier chapters also presented opportunities for further research in this field. The computational times are very strongly correlated to the fitness function and one could try using a neural network architecture to predict fitness instead of calculating them, given matrixes of subspaces for the extreme points and a matrix of item to be inserted in its varying orientations. Generally research on different ways of representing the spaces and items could increase the efficiency of the algorithm. More importantly whether to consider all of the extreme points (so using the “Fill” property) or to limit the lowest point to be considered for the packing related to the skyline could not only decrease computational times per iteration and also increase sensitivity to the sequence. Such a consideration would make the construction algorithm take less time to finish packing, giving any metaheuristic more time to try out more combinations on the sequence.

Beyond the computational efficiency per iteration and sequence modifications, an implementation for more dimensions, so 3 geometric and more constraints regarding packing, would be interesting. Strip packing problems are relevant for the efficient use of commodities such as fabric and metal and representing the irregular items in ways that can be easily packed could be a challenge to tackle with great decrease in wasted material. Unloading constraints come into play with the introduction of routing, and 2l-3l CVRP are other interesting problems to see how this algorithm could perform as it tries to pack well with the given sequence, possibly allowing more sequence changes regarding routing as opposed to other heuristics where the packing could be considered infeasible.

In this thesis, an approach for packing was utilized that tried to accommodate a given fixed sequence and this approach succeeded in finding competitive results. Improved implementations or modifications of this construction heuristic could yield even better results and work hand in hand with sequence modification. In conclusion, this research field and this problem are open to fresh construction and improvement methods beyond simple sequence modification and some examples of this can be found in the literature mentioned in Chapter 1 and the References. This thesis proves that well fitting construction methods could achieve comparably good results to the best performing metaheuristics and it is worthwhile to look into other alternative heuristics and efficient implementations. All the code and the solution files created in the scope of this thesis with an additional proprietary visualization tool for the solutions can be found in the following repository: tandoganbaris/Masterarbeit_library2 (github.com)

7 References

1. Baker, B. S., Coffman, Jr., E. G., & Rivest, R. L. (1980). Orthogonal Packings in Two Dimensions. SIAM Journal on Computing, 9(4), 846–855. https://doi.org/10.1137/0209064
2. Chazelle. (1983). The Bottom-Left Bin-packing heuristic: An Efficient Implementation. IEEE Transactions on Computers, C–32(8), 697–707. https://doi.org/10.1109/tc.1983.1676307
3. Berkey, J. O., & Wang, P. Y. (1987). Two-Dimensional Finite Bin-Packing Algorithms. The Journal of the Operational Research Society, 38(5), 423. https://doi.org/10.2307/2582731
4. Scheithauer Guntram. (1999). Equivalence and Dominance for Problems of Optimal Packing of Rectangles.
5. 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. https://doi.org/10.1287/ijoc.11.4.345
6. Mumford-Valenzuela, C. L., Vick, J., & Wang, P. Y. (2003). Heuristics for Large Strip Packing Problems with Guillotine Patterns: An Empirical Study. In Springer eBooks (pp. 501–522). https://doi.org/10.1007/978-1-4757-4137-7_24
7. David, Jesus & Calderón, Jose & Cabrera, Rayco & Moreno-Pérez, José & Moreno-Vega, J.. (2004). GRASP-VNS hybrid for the Strip Packing Problem.. 79-90.
8. Burke, E. K., Kendall, G., & Whitwell, G. (2004). A new placement heuristic for the orthogonal Stock-Cutting problem. Operations Research, 52(4), 655–671. https://doi.org/10.1287/opre.1040.0109
9. Karabulut, K., & Inceoglu, M. M. (2004). A Hybrid Genetic Algorithm for Packing in 3D with Deepest Bottom Left with Fill Method. In Lecture Notes in Computer Science (pp. 441–450). https://doi.org/10.1007/978-3-540-30198-1_45
10. Cordeau, Jean-François & Gendreau, Michel & Hertz, Alain & Laporte, Gilbert & Sormany, Jean-Sylvain. (2005). New Heuristics for the Vehicle Routing Problem. 10.1007/0-387-24977-X_9.
11. Bortfeldt, A. (2006). A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. European Journal of Operational Research, 172(3), 814–837. https://doi.org/10.1016/j.ejor.2004.11.016
12. Alvarez-Valdés, R., Parreño, F., & Tamarit, J. M. (2008). Reactive GRASP for the strip-packing problem. Computers & Operations Research, 35(4), 1065–1083. https://doi.org/10.1016/j.cor.2006.07.004
13. Crainic, T. G., Perboli, G., & Tadei, R. (2008). Extreme Point-Based heuristics for Three-Dimensional bin packing. Informs Journal on Computing, 20(3), 368–384. https://doi.org/10.1287/ijoc.1070.0250
14. Fuellerer, Guenther & Doerner, Karl & Hartl, Richard & Iori, Manuel. (2009). Ant colony optimization for the two-dimensional loading vehicle routing problem. Computers & Operations Research. 36. 655-673. 10.1016/j.cor.2007.10.021.
15. Iori, Manuel & Salazar González, Juan José & Vigo, Daniele. (2007). An Exact Approach for the Vehicle Routing Problem with Two-Dimensional Loading Constraints. Transportation Science. 41. 253-264. 10.1287/trsc.1060.0165.
16. Neveu, B., Trombettoni, G., Araya, I., & Riff, M. C. (2008). A STRIP PACKING SOLVING METHOD USING AN INCREMENTAL MOVE BASED ON MAXIMAL HOLES. International Journal on Artificial Intelligence Tools, 17(05), 881–901. https://doi.org/10.1142/s0218213008004205
17. Gendreau, Michel & Iori, Manuel & Laporte, Gilbert & Martello, Silvaro. (2008). A Tabu Search heuristic for the vehicle routing problem with two-dimensional loading constraints. Networks. 51. 4 - 18. 10.1002/net.20192.

18. Imahori, S., & Yagiura, M. (2010). The best-fit heuristic for the rectangular strip packing problem: An efficient implementation and the worst-case approximation ratio. Computers & Operations Research, 37(2), 325–333. https://doi.org/10.1016/j.cor.2009.05.008
19. Asik, O. B., & Özcan, E. (2009). Bidirectional best-fit heuristic for orthogonal rectangular strip packing. Annals of Operations Research, 172(1), 405–427. https://doi.org/10.1007/s10479-009-0642-0
20. Ortmann, F., Ntene, N., & Van Vuuren, J. (2010). New and improved level heuristics for the rectangular strip packing and variable-sized bin packing problems. European Journal of Operational Research, 203(2), 306–315. https://doi.org/10.1016/j.ejor.2009.07.024
21. Burke, E. K., Kendall, G., & Whitwell, G. (2009). A simulated annealing enhancement of the Best-Fit heuristic for the orthogonal Stock-Cutting problem. Informs Journal on Computing, 21(3), 505–516. https://doi.org/10.1287/ijoc.1080.0306
22. Fuellerer, Guenther & Doerner, Karl & Hartl, Richard & Iori, Manuel. (2010). Metaheuristics for vehicle routing problems with three-dimensional loading constraints. European Journal of Operational Research. 201. 751-759. 10.1016/j.ejor.2009.03.046.
23. Duhamel, Christophe & Lacomme, Philippe & Quilliot, Alain & Toussaint, Hélène. (2009). 2L-CVRP: A GRASP resolution scheme based on RCPSP. 10.1109/ICCIE.2009.5223772.
24. Zachariadis, E. E., Tarantilis, C. D., & Kiranoudis, C. T. (2009). A Guided Tabu Search for the Vehicle Routing Problem with two-dimensional loading constraints. European Journal of Operational Research, 195(3), 729–743. https://doi.org/10.1016/j.ejor.2007.05.058
25. Leung, S. C. H., Zhang, D., Zhou, C., & Wu, T. (2012). A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem. Computers & Operations Research, 39(1), 64–73. https://doi.org/10.1016/j.cor.2010.10.022
26. Allen, S. D., Burke, E. K., & Kendall, G. (2011). A hybrid placement strategy for the three-dimensional strip packing problem. European Journal of Operational Research, 209(3), 219–227. https://doi.org/10.1016/j.ejor.2010.09.023
27. Duhamel, Christophe & Lacomme, Philippe & Quilliot, Alain & Toussaint, Hélène. (2011). A multi-start evolutionary local search for the two-dimensional loading capacitated vehicle routing problem. Computers & Operations Research. 38. 617-640. 10.1016/j.cor.2010.08.017.
28. Strodl, Johannes & Doerner, Karl & Tricoire, Fabien & Hartl, Richard. (2010). On Index Structures in Hybrid Metaheuristics for Routing Problems with Hard Feasibility Checks: An Application to the 2-Dimensional Loading Vehicle Routing Problem. 160-173. 10.1007/978-3-642-16054-7_12.
29. Leung, Stephen & Zheng, Jiemin & Zhang, Defu & Zhou, Xiyue. (2010). Simulated annealing for the vehicle routing problem with two-dimensional loading constraints. Flexible Services and Manufacturing Journal. 22. 61-82. 10.1007/s10696-010-9061-4.
30. Leung, Stephen & Zhou, Xiyue & Zhang, Defu & Zheng, Jiemin. (2011). Extended guided Tabu search and a new packing algorithm for the two-dimensional loading vehicle routing problem. Computers & OR. 38. 205-215. 10.1016/j.cor.2010.04.013.
31. Leung, S. C. H., Zhang, D., & Sim, K. M. (2011). A two-stage intelligent search algorithm for the two-dimensional strip packing problem. European Journal of Operational Research, 215(1), 57–69. https://doi.org/10.1016/j.ejor.2011.06.002
32. Lijun, Wei & Lim, Andrew & Zhu, Wenbin. (2011). A Skyline-Based Heuristic for the 2D Rectangular Strip Packing Problem. 6704. 286-295. 10.1007/978-3-642-21827-9_29.
33. Da Silveira, J. L. M., Miyazawa, F. K., & Xavier, E. C. (2013). Heuristics for the strip packing problem with unloading constraints. Computers & Operations Research, 40(4), 991–1003. https://doi.org/10.1016/j.cor.2012.11.003
34. Leung, S. C. H., Zhang, Z., Zhang, D., Hua, X., & Lim, M. K. (2013). A meta-heuristic algorithm for heterogeneous fleet vehicle routing problems with two-dimensional loading

constraints. European Journal of Operational Research, 225(2), 199–210. https://doi.org/10.1016/j.ejor.2012.09.023
35. Shen, Yuxi & Murata, Tomohiro. (2012). Pick-up Scheduling of Two-dimensional Loading in Vehicle Routing Problem by using GA. Lecture Notes in Engineering and Computer Science. 2196. 1532-1537.
36. Martinez, L. & Amaya, Ciro. (2012). A vehicle routing problem with multi-trips and time windows for circular items. J Oper Res Soc. 64. 10.1057/jors.2012.128.
37. Yang, Shuangyuan & Han, Shuihua & Ye, Weiguo. (2013). A simple randomized algorithm for two-dimensional strip packing. Computers & Operations Research. 40. 10.1016/j.cor.2012.05.001.
38. Özcan, E., Zhang, K., & Drake, J. H. (2013). Bidirectional best-fit heuristic considering compound placement for two dimensional orthogonal rectangular strip packing. Expert Systems With Applications, 40(10), 4035–4043. https://doi.org/10.1016/j.eswa.2013.01.005
39. Zhang, Defu & Lijun, Wei & Leung, Stephen & Chen, Qingshan. (2013). A Binary Search Heuristic Algorithm Based on Randomized Local Search for the Rectangular Strip Packing Problem. INFORMS Journal on Computing 25 (2) (2013) 332-345. 25. 332-345. 10.1287/ijoc.1120.0505.
40. Bin, Wu & Hong, Cai & Zhi-yong, Cui. (2013). Artificial bee colony algorithm for two-dimensional loading capacitated vehicle routing problem. International Conference on Management Science and Engineering - Annual Conference Proceedings. 406-412. 10.1109/ICMSE.2013.6586313.
41. Côté, J., Gendreau, M., & Potvin, J. (2014). An Exact Algorithm for the Two-Dimensional Orthogonal Packing Problem with Unloading Constraints. Operations Research, 62(5), 1126–1141. https://doi.org/10.1287/opre.2014.1307
42. Dominguez, Oscar & Juan, Angel & Barrios, Barry & Faulin, Javier & Agustín, A.. (2014). Using biased randomization for solving the two-dimensional loading vehicle routing problem with heterogeneous fleet. Annals of Operations Research. 236. 10.1007/s10479-014-1551-4.
43. Chen, Bili & Wang, Yong & Yang, Shuangyuan. (2015). A Hybrid Demon Algorithm for the Two-Dimensional Orthogonal Strip Packing Problem. Mathematical Problems in Engineering. 2015. 1-14. 10.1155/2015/541931.
44. Côté, Jean-François & Guastaroba, Gianfranco & Speranza, M.Grazia. (2016). The Value of Integrating Loading and Routing. European Journal of Operational Research. 257. 10.1016/j.ejor.2016.06.072.
45. Lijun, Wei & Zhang, Zhenzhen & Zhang, Defu & Lim, Andrew. (2015). A variable neighborhood search for the capacitated vehicle routing problem with two-dimensional loading constraints. European Journal of Operational Research. 243. 798-814. 10.1016/j.ejor.2014.12.048.
46. Lijun, Wei & Zhang, Zhenzhen & Lim, Andrew. (2014). An evolutionary local search for the capacitated vehicle routing problem minimizing fuel consumption under three-dimensional loading constraints. 2014 10th International Conference on Natural Computation, ICNC 2014. 10.1109/ICNC.2014.6975835.
47. Hokama, Pedro & Miyazawa, Flavio & Xavier, Eduardo. (2015). A Branch-and-Cut Approach for the Vehicle Routing P roblem with Two-dimensional Loading Constraints. Expert Systems with Applications. 47. 10.1016/j.eswa.2015.10.013.
48. Oliveira, José & Neuenfeldt Júnior, Alvaro & Silva, Elsa & Carravilla, Maria. (2016). A survey on heuristics for the two-dimensional rectangular strip packing problem. Pesquisa Operacional. 36. 197-226. 10.1590/0101-7438.2016.036.02.0197.
49. Domínguez, O. F. C., Guimarans, D., Juan, A. A., & De La Nuez, I. (2016). A Biased-Randomised Large Neighbourhood Search for the two-dimensional Vehicle Routing

Problem with Backhauls. European Journal of Operational Research, 255(2), 442–462. https://doi.org/10.1016/j.ejor.2016.05.002
50. Lijun, Wei & Qin, Hu & Cheang, Brenda & Xu, Xianhao. (2014). An efficient intelligent search algorithm for the two-dimensional rectangular strip packing problem. International Transactions in Operational Research. 23. 10.1111/itor.12138.
51. Babaoglu, I. (2017). Solving 2D strip packing problem using fruit fly optimization algorithm. Procedia Computer Science, 111, 52–57. https://doi.org/10.1016/j.procs.2017.06.009
52. Lam, Thuan & Ho, Viet & Logofătu, Doina & Badica, Costin. (2017). Considerations on using genetic algorithms for the 2D bin packing problem: A general model and detected difficulties. 303-308. 10.1109/ICSTCC.2017.8107051.
53. Zhang, D., Dong, R., Si, Y., Ye, F., & Cai, Q. (2018). A hybrid swarm algorithm based on ABC and AIS for 2L-HFCVRP. Applied Soft Computing, 64, 468–479. https://doi.org/10.1016/j.asoc.2017.12.012
54. Chen, Zhen & Chen, Jianli. (2018). An Effective Corner Increment-Based Algorithm for the Two-Dimensional Strip Packing Problem. IEEE Access. PP. 1-1. 10.1109/ACCESS.2018.2882823.
55. Wei, L., Zhang, Z., Zhang, D., & Leung, S. C. H. (2018). A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. European Journal of Operational Research, 265(3), 843–859. https://doi.org/10.1016/j.ejor.2017.08.035
56. Chen, Mao & Wu, Chao & Tang, Xiangyang & Peng, Xicheng & Zeng, Zhizhong & Liu, Sanya. (2019). An efficient deterministic heuristic algorithm for the rectangular packing problem. Computers & Industrial Engineering. 137. 106097. 10.1016/j.cie.2019.106097.
57. Wei, L., Wang, Y., Cheng, H., & Huang, J. (2019). An open space based heuristic for the 2D strip packing problem with unloading constraints. Applied Mathematical Modelling, 70, 67–81. https://doi.org/10.1016/j.apm.2019.01.022
58. Rovensky Daniel. (2019). GRASP for the two-dimensional strip packing problem.
59. Rakotonirainy, Rosephine & Vuuren, Jan. (2020). Improved metaheuristics for the two-dimensional strip packing problem. Applied Soft Computing. 92. 106268. 10.1016/j.asoc.2020.106268.
60. Grandcolas, Stéphane & Pain-Barre, Cyril. (2022). A hybrid metaheuristic for the two-dimensional strip packing problem. Annals of Operations Research. 309. 10.1007/s10479-021-04226-6.
61. Júnior, A. L. N., Silva, E., Francescatto, M. B., Rosa, C. B., & Siluk, J. C. M. (2022). The rectangular two-dimensional strip packing problem real-life practical constraints: A bibliometric overview. Computers & Operations Research, 137, 105521. https://doi.org/10.1016/j.cor.2021.105521
62. Guo, P., & Jiang, M. (2022). TSMSA: a 2DSPP algorithm with multi-strategy rectangle selection. The Journal of Supercomputing. https://doi.org/10.1007/s11227-022-04350-5
63. Piechowiak, K., Drozdowski, M., & Sanlaville, E. (2022). Framework of algorithm portfolios for strip packing problem. Computers & Industrial Engineering, 172, 108538. https://doi.org/10.1016/j.cie.2022.108538
64. Oliveira, Ó., Gamboa, D., & Silva, E. (2022). An introduction to the two‐dimensional rectangular cutting and packing problem. International Transactions in Operational Research. https://doi.org/10.1111/itor.13236
65. Júnior, A. L. N., Siluk, J. C. M., Francescatto, M. B., Stieler, G., & Disconzi, D. (2023). A framework to select heuristics for the rectangular two-dimensional strip packing problem. Expert Systems With Applications, 213, 119202. https://doi.org/10.1016/j.eswa.2022.119202

66. Oviedo-Salas, E., Terán-Villanueva, J. D., Martínez, S. I., Santiago, A., Ponce-Flores, M., Laria-Menchaca, J., Rocha, J. a. C., & Treviño-Berrones, M. G. (2022). GRASP Optimization for the Strip Packing Problem with Flags, Waste Functions, and an Improved Restricted Candidate List. Applied Sciences, 12(4), 1965. https://doi.org/10.3390/app12041965
67. Xiang Yi, Zhang & Chen, Lu & Gendreau, Michel & Langevin, André. (2022). A branch-and-cut algorithm for the vehicle routing problem with two-dimensional loading constraints. European Journal of Operational Research. 302. 10.1016/j.ejor.2021.12.050.
68. Krebs, Corinna & Ehmke, Jan & Koch, Henriette. (2022). Effective loading in combined vehicle routing and container loading problems. Computers & Operations Research. 149. 105988. 10.1016/j.cor.2022.105988.
69. Candido, L. C. X., & De Souza, L. V. (2022). Mathematical Model and Simulated Annealing Algorithm for the Two-Dimensional Loading Heterogeneous Fixed Fleet Vehicle Routing Problem. Mathematical Problems in Engineering, 2022, 1–19. https://doi.org/10.1155/2022/6012105
70. De Queiroz, T. A., De Queiroz, T. A., & De Toledo, F. M. B. (2022). Integer Formulations for the Integrated Vehicle Routing Problem With Two-dimensional Packing Constraints. Pesquisa Operacional, 42. https://doi.org/10.1590/0101-7438.2022.042.00248686
71. Zhang, X., Lu, C., Gendreau, M., & Langevin, A. (2022). Learning-Based Branch-and-Price Algorithms for the Vehicle Routing Problem with Time Windows and Two-Dimensional Loading Constraints. Informs Journal on Computing, 34(3), 1419–1436. https://doi.org/10.1287/ijoc.2021.1110
72. Que, Q., Yang, F., & Zhang, D. (2023). Solving 3D packing problem using Transformer network and reinforcement learning. Expert Systems With Applications, 214, 119153. https://doi.org/10.1016/j.eswa.2022.119153

APPENDIX

Name	n	Optimal	EPSP-SA	Time	%EPSP-SA	GRASP	SVC	ISA	HDA	SRA	CIBA	HHA	TSMSA
n1a	17	200	211	0.082	5.5	0	1	0	0	0	0	0	0
n1b	17	200	208	0.085	4	4.5	0	5.6	0	0	0	0	0
n1c	17	200	209	0.076	4.5	0	0	0	0	0	0	0	0
n1d	17	200	213	0.083	6.5	0	0	0	0	0	0	0	0
n1e	17	200	210	0.077	5	0	0	0	0	0	0	0	0
n2a	25	200	209	0.093	4.5	3	2.5	2	0.7	0	0	0	0.7
n2b	25	200	209	0.097	4.5	3	4.5	4.7	5	0	0	0	0
n2c	25	200	212	0.106	6	4	4.5	4.3	3.2	1.5	1.5	0	1.8
n2d	25	200	208	0.091	4	4.5	3.5	3.9	3	2.9	1.8	0	0.3
n2e	25	200	211	0.092	5.5	3	2.5	3.3	3.1	2.6	1.5	0	0.8
n3a	29	200	210	0.148	5	4.5	4	3.1	3.2	0	3.1	0	1.1
n3b	29	200	209	0.084	4.5	4	3.5	4.5	4.5	4.3	3.5	1	3.8
n3c	29	200	208	0.091	4	2.5	3.5	3.1	2.5	0	2.9	2.3	2.8
n3d	29	200	210	0.104	5	3.5	4	2.2	2.3	2	0.6	1.4	0
n3e	29	200	207	0.092	3.5	3.5	3.5	4	4.1	3.6	2	0	0
n4a	49	200	207	0.154	3.5	3	2.5	3	3	2.6	2.2	2	2
n4b	49	200	206	0.168	3	3.5	2.5	2.5	2.3	2.2	2.6	2.4	2.6
n4c	49	200	206	0.154	3	2.5	2.5	3	2.7	2.6	2.2	1.7	2
n4d	49	200	207	0.133	3.5	3	2.5	2.4	2.5	2.6	2	2.1	2.2
n4e	49	200	206	0.109	3	2.5	2.5	3	3.1	3.3	1.8	2	2.6
n5a	73	200	205	0.144	2.5	2.5	2	2.6	2.8	2.4	1.8	1.5	1.7

n5b	73	200	204	0.171	2	2	2	1.8	1.6	1.9	1.6	1.6	1.4
n5c	73	200	205	0.14	2.5	3	2	2.2	2.3	2.1	1.9	1.7	1.8
n5d	73	200	205	0.138	2.5	2	2.5	2.5	2.3	2.3	2	2.2	1.8
n5e	73	200	205	0.151	2.5	3	2.5	2.3	2.5	2.4	1.8	1.7	1.7
n6a	97	200	204	0.237	2	2	1.5	1.4	1.3	1.5	1.2	1.5	1.3
n6b	97	200	203	0.244	1.5	2	2	1.5	1.5	1.5	1.2	1.3	1
n6c	97	200	203	0.241	1.5	2	2	1.8	1.6	1.7	1.2	1.3	1
n6d	97	200	204	0.285	2	2.1	1	1.9	1.5	1.5	1.5	1.5	1.3
n6e	97	200	204	0.258	2	2	1.5	1.8	1.5	1.4	1.3	1.2	1.1
n7a	199	200	202	0.959	1	1	1	0.5	0.5	0.5	0.5	0.5	0.5
n7b	199	200	201	1.08	0.5	1.5	1	1	0.5	0.5	0.5	0.5	0.5
n7c	199	200	202	0.839	1	1.5	1	1	0.5	0.5	0.5	0.5	0.5
n7d	199	200	202	0.886	1	1.5	1	1	0.5	0.5	0.5	0.5	0.5
n7e	199	200	202	1.053	1	1.5	1	1	0.5	0.5	0.5	0.5	0.5
t1a	17	200	210	0.073	5	0	0	0	0	0	0	0	0
t1b	17	200	213	0.077	6.5	0	5.5	0	0	0	0	0	0
t1c	17	200	213	0.085	6.5	0	5	0	0	0	0	0	0
t1d	17	200	209	0.08	4.5	0	0	5.9	0	0	0	0	0
t1e	17	200	216	0.074	8	0	4.5	0	0	0	0	0	0
t2a	25	200	209	0.096	4.5	2	3.5	3.5	3.2	3	1.8	0	0
t2b	25	200	206	0.104	3	4	2.5	3.5	2.9	0	1.1	0	0
t2c	25	200	210	0.118	5	4	3	3	3.7	3	1.4	0.6	0
t2d	25	200	211	0.093	5.5	3	3.5	4.7	2.2	0	1.3	0	2
t2e	25	200	210	0.084	5	3	3.5	3.7	3	2.5	1	0	0
t3a	29	200	210	0.093	5	3.5	4	4.5	4.5	0	1.3	1.2	0

t3b	29	200	208	0.09	4	4.5	3.5	4.1	4	3	3.6	2.8	2.3
t3c	29	200	208	0.085	4	3	3.5	3.3	3.2	2.5	0.8	0	0
t3d	29	200	207	0.105	3.5	3.5	4	3.2	3.2	0	0.9	1.6	1.7
t3e	29	200	207	0.112	3.5	4	3	2.5	2.5	2.5	3.6	1	3.8
t4a	49	200	206	0.13	3	2.5	2.5	2.5	2.3	2.5	2.3	2	2.2
t4b	49	200	205	0.13	2.5	2.5	2.5	3.1	3	3	2.2	1.6	2.1
t4c	49	200	205	0.119	2.5	3	2.5	2.5	2.3	1.5	1.8	2.1	2.3
t4d	49	200	207	0.124	3.5	3	2.5	2.8	2.7	1	2.2	2	2.2
t4e	49	200	206	0.117	3	3.5	2.5	2.6	2.8	2	2.6	2.3	2.8
t5a	73	200	203	0.237	1.5	3	2	2.2	2.3	2	1.8	1.7	1.7
t5b	73	200	203	0.158	1.5	2	2	2	2.2	1.5	1.5	1.2	1.3
t5c	73	200	205	0.153	2.5	2.5	2	2.5	2.8	2	1.8	1.5	1.7
t5d	73	200	205	0.171	2.5	2	2.5	2.5	2.3	2	2	2.1	1.9
t5e	73	200	205	0.186	2.5	2	2	2	2.3	2	1.6	2.3	1.5
t6a	97	200	202	0.245	1	2	2	1.6	1.8	1.5	1.3	1.2	1
t6b	97	200	203	0.241	1.5	2	1	1.7	1.6	1.5	1.1	1.3	1.1
t6c	97	200	203	0.202	1.5	2	2	1.5	1.3	1.5	1.5	1.2	1.2
t6d	97	200	203	0.257	1.5	2	2	1.8	1.8	1.5	1.2	1.3	1.1
t6e	97	200	203	0.245	1.5	2.5	2	1.8	1.8	1.5	1.4	1.3	1
t7a	199	200	202	1.012	1	1.5	0.5	0.6	0.5	0.5	0.5	0.5	0.5
t7b	199	200	202	1.333	1	1.5	1	0.5	0.5	0.5	0.5	0.5	0.5
t7c	199	200	202	0.754	1	2	1	0.5	0.5	0.5	0.5	0.5	0.5
t7d	199	200	203	0.899	1.5	1	1	1	0.5	0.5	0.5	0.5	0.5
t7e	199	200	203	0.927	1.5	1.5	1	0.8	0.5	0.5	0.5	0.5	0.5
Avg					3.214	2.322	2.271	2.247	1.925	1.391	1.304	0.96771	1.095

Instances N and T (Figure 18)

Name	n	W	LB	SA-Avg	EPSP-SA	Time	%EPSP-SA	GRASP	SVC	ISA	HDA	SRA	CIBA	HHA	TSMSA
cgcut1	16	10	23	23.3	23	0.039	1.3	0	0	0	4.3	4.4	0	0	0
cgcut2	23	70	63	66.5	66	0.044	5.56	3.2	3.2	3.2	3.2	3.2	3.2	3.2	3.2
cgcut3	62	70	636	655	655	0.074	2.99	3.9	3.9	3.8	4.1	4	3.8	3.8	3.8
gcut1	10	250	1016	780	779	0.041	-23.23	0	0	0	0	0	0	0	0
gcut2	20	250	1133	1210.33	1212	0.05	6.83	5.1	4.8	4.8	4.8	4.8	4.8	4.8	4.8
gcut3	30	250	1803	1768	1767	0.061	-1.94	0	0	0	0	0	0	0	0
gcut4	50	250	2934	3073.5	3069	0.099	4.75	2.3	2.8	2.6	2.3	2.4	2.6	2.8	2.7
gcut5	10	500	1172	1257	1257	0.091	7.25	8.6	8.6	8.6	8.6	8.6	8.6	8.6	8.6
gcut6	20	500	2514	2718.67	2668	0.082	8.14	4.5	4.7	4.7	4.6	4.5	4.5	4.3	4.3
gcut7	30	500	4641	4520.33	4452	0.115	-2.6	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
gcut8	50	500	5703	5999.5	5970	0.107	5.2	3.7	3	3.3	3.2	3.2	3.1	3.1	3.1
gcut9	10	1000	2022	2242	2236	0.032	10.88	14.6	14.6	14.6	14.6	14.6	14.6	14.6	10.8
gcut10	20	1000	5356	5634	5634	0.048	5.19	11.4	11.4	11.4	11.4	11.4	11.4	11.4	19.4
gcut11	30	1000	6537	7109	7072	0.04	8.75	5.5	5.4	5.3	5.3	5	5.2	5.2	18.3
gcut12	50	1000	12,522	13280	13280	0.075	6.05	17.3	17.3	17.3	17.3	17.3	17.3	17.3	5.2
gcut13	32	3000	4772	4766.5	4730	0.084	-0.12	4.7	4.3	4.1	4	4.2	4.2	3.3	5
ngcut1	10	10	23	20	20	0.081	-13.04	0	0	0	0	0	0	0	0
ngcut2	17	10	30	28	28	0.086	-6.67	0	0	3.3	3.3	3.3	0	0	0
ngcut3	21	10	28	28	28	0.095	0	0	0	0	0	0	0	0	0
ngcut4	7	10	20	20	20	0.082	0	0	0	0	0	0	0	0	0
ngcut5	14	10	36	36	36	0.08	0	0	0	0	0	0	0	0	0
ngcut6	15	10	29	30	30	0.135	3.45	6.9	6.9	6.9	6.9	6.9	6.9	6.9	6.9

ngcut7	8	20	20	20	20	0.095	0	0	0	0	0	0	0	0	0
ngcut8	13	20	32	34	34	0.083	6.25	3.1	6.3	6.3	6.3	6.3	6.3	6.3	6.3
ngcut9	18	20	49	50	50	0.08	2.04	2	4.1	6.1	4.1	4.1	4.1	2	2
ngcut10	13	30	80	62	62	0.074	-22.5	0	0	0	0	0	0	0	0
ngcut11	15	30	50	53	53	0.074	6	4	4	4	4	4	4	4	4
ngcut12	22	30	87	81	81	0.081	-6.9	0	0	0	0	0	0	0	0
beng1	20	25	30	30	30	0.03	0	0	0	3.3	2.3	3.3	3.3	0	0
beng2	40	25	57	57	57	0.055	0	0	0	0	0	0	0	0	0
beng3	60	25	84	84	84	0.079	0	0	0	0	0	0	0	0	0
beng4	80	25	107	107	107	0.132	0	0	0	0	0	0	0	0	0
beng5	100	25	134	134	134	0.153	0	0	0	0	0	0	0	0	0
beng6	40	40	36	36	36	0.141	0	0	0	0	0	0	0	0	0
beng7	80	40	67	67	67	0.128	0	0	0	0	0	0	0	0	0
beng8	120	40	101	101	101	0.167	0	0	0	0	0	0	0	0	0
beng9	160	40	126	126	126	0.258	0	0	0	0	0	0	0	0	0
beng10	200	40	156	156	156	0.483	0	0	0	0	0	0	0	0	0
Avg							2.385	2.682	2.8	3.018	3.045	3.068	2.868	2.703	2.882

Instances non-zero Waste (Figure 19)