UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A STUDY OF BI-SPACE SEARCH FOR BIN PACKING PROBLEMS

by

Derrick Beckedahl

Submitted in fulfillment of the requirements for the degree

Doctor of Philosophy (Computer Science)

in the

Department of Computer Science

Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

May 2024

ABSTRACT

Traditionally, search techniques explore a single space, namely the solution space, to find a solution to a discrete optimisation problem. However, as the field has developed, the effectiveness of working in alternative spaces (such as the heuristic space) has been demonstrated. In addition, the most effective search techniques are computationally expensive. More recently, exploring more than one space to solve a problem has been investigated. This research has involved searching in the heuristic and solution space sequentially or alternating the search in each space. The first aim of this study is the introduction of the concept of concurrent bi-space search (CBS) which involves searching in both the solution and heuristic spaces concurrently. It is anticipated that this will be more effective than searching a single space or performing a search in both spaces sequentially. Previous work has shown that searching in alternative spaces, like the heuristic space, is computationally expensive. Furthermore, in an attempt to improve the quality of solutions found, computationally expensive approaches are used to explore the solution space. Thus, a secondary aim of this study is to use a search technique that is computationally cheap to concurrently search the solution and heuristic spaces. It is hypothesised that exploring both spaces concurrently will eliminate the need to use computationally expensive techniques to explore the solution space to produce solutions of effective quality.

While the concept of CBS can be applied to any discrete optimisation problem, this study is restricted to packing problems, specifically the one-two- and three-dimensional bin packing problems (1BPP, 2BPP and 3BPP). The higher dimension BPPs are chosen to investigate the scalability of the approach. A simple local search is used to independently search the heuristic (HSS) and solution (SSS) spaces in order to obtain a baseline against which to compare the CBS approach which also employs a local search to concurrently search the heuristic and solution spaces.

Performance comparison of the three approaches (CBS, HSS and SSS) is conducted using three different performance metrics, namely the number of bins, a measure of the total wasted space across the bins, i.e. the packing efficiency, and the computational time. For all three problem domains (1BPP, 2BPP and 3BPP) CBS outperforms both HSS and SSS in terms of the number of bins and the amount of wasted space. However, SSS has lower runtimes, with CBS having lower runtimes than HSS. These results are found to be statistically significant for the majority of the problem instances.

When compared to previous bi-space search approaches, CBS is found to both produce better quality solutions and have faster average runtimes. The CBS approach is also compared to state-of-the-art techniques for 1BPP, 2BPP and 3BPP. The CBS approach does not outperform the state-of-the-art techniques for the simpler 1BPP, but is found to be scalable to the more difficult 2BPP and 3BPP, having comparable performance to the state-of-the-art techniques and in some cases outperforming them.

PUBLICATIONS

The following publications are associated with the research presented in this thesis:

1. D. Beckedahl and N. Pillay. A study of bi-space search for solving the one-dimensional bin packing problem. In L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz, and J. Zurada, editors, Artificial Intelligence and Soft Computing, pages 277–289, Cham, 2020. Springer International Publishing. ISBN 978-3-030-61534-5.
2. D. Beckedahl and N. Pillay. Bi-space search: Optimizing the hybridization of search spaces in solving the one dimensional bin packing problem. In L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz, and J. Zurada, editors, Artificial Intelligence and Soft Computing, pages 206–217, Cham, 2023. Springer International Publishing. ISBN 978-3-031-23480-4.
3. D. Beckedahl, N. Pillay and T. Nyathi. An Improved Bi-Space Search for Bin Packing Problems. OR Spectrum. *under review*
4. D. Beckedahl, N. Pillay and T. Nyathi. Concurrent Bi-Space Search for Bin Packing Problems. Annals of Operations Research. *under review*

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Prof. Nelishia Pillay, for her guidance and support during this process as well as her endless patience with me over the years. Thank you also to my co-supervisor, Dr. Thambo Nyathi, for his assistance and feedback during this process, and for encouraging me to persevere when I needed it most.

To my grandparents for all your support and encouragement. To Pat, for all the good times, helping me escape when things got too much, and just generally keeping me sane over the years. Thank you to my parents for providing me with the tools I needed to get to this point in my life, for the endless support and encouragement, and for always believing in me.

Finally, I would like to acknowledge the financial support from the Multichoice Research Chair in Machine Learning at the University of Pretoria.

LIST OF ABBREVIATIONS

1BPP	**1**-dimensional **B**in **P**acking **P**roblem
2BPP	**2**-dimensional **B**in **P**acking **P**roblem
3BPP	**3**-dimensional **B**in **P**acking **P**roblem
AILTA	**A**daptive **I**teration **L**imited **T**hreshold **A**ccepting
BISON	**Bi**n packing **So**lutio**n** procedure
BPP	**B**in **P**acking **P**roblem
BRKGA	**B**iased **R**andom-**K**ey **G**enetic **A**lgorithm
BSS	**B**i-**S**pace **S**earch
CBS	**C**oncurrent **B**i-space **S**earch
CNS-BP	**C**onsistent **N**eighbourhood **S**earch for **B**in **P**acking
CSA	**C**oncurrent **S**earch **A**pproach
CSP	**C**utting **S**tock **P**roblem
EA	**E**volutionary **A**lgorithm
GAHH	**G**enetic **A**lgorithm **H**yper-**H**euristic
GA	**G**enetic **A**lgorithm
GE	**G**rammatical **E**volution
GGA-CGT	**G**rouping **G**enetic **A**lgorithm with **C**ontrolled **G**ene **T**ransmission
GGA	**G**rouping **G**enetic **A**lgorithm
GHH	**G**raph-based **H**yper-**H**euristic
GLS	**G**uided **L**ocal **S**earch
GPHH	**G**enetic **P**rogramming **H**yper-**H**euristic
GP	**G**enetic **P**rogramming
GRASP	**G**reedy **R**andomized **A**daptive **S**earch **P**rocedure
GVND	**G**RASP with **VND**
HBP	**H**euristic for **B**in **P**acking
HI-BP	**H**ybrid **I**mprovement procedure for **B**in **P**acking

LIST OF ABBREVIATIONS (CONT.)

HSS	**H**euristic **S**pace **S**earch
ISA	**I**nterleaving **S**earch **A**pproach
KP	**K**napsack **P**roblem
LLH	**L**ow-**L**evel **H**euristic
LS	**L**ocal **S**earch
MA	**M**emetic **A**lgorithm
MBS	**M**inimum **B**in **S**lack
MTRP	**M**artello and **T**oth **R**eduction **P**rocedure
SAHH	**S**imulated **A**nnealing **H**yper-**H**euristic
SAW	**S**ufficient **A**verage **W**eight
SCH	**S**et-**C**overing-based **H**euristic
SNGP	**S**ingle **N**ode **G**enetic **P**rogramming
SPP	**S**trip **P**acking **P**roblem
SSA	**S**equential **S**earch **A**pproach
SSS	**S**olution **S**pace **S**earch
TS2	**T**abu Search for **Two**-dimensional bin packing
TS2PACK	(**T**abu **S**earch)$^{\mathbf{2}}$ for bin **Pack**ing
TS3	**T**abu **S**earch for **Three**-dimensional bin packing
VND	**V**ariable **N**eighbourhood **D**escent
VNS	**V**ariable **N**eighbourhood **S**earch

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

The process of search involves exploring a search space, which is a set of states and operators which can be applied to those states [1, 2]. Search techniques typically explore a single search space [2]. Traditionally this has been a space consisting of candidate solutions, this space is commonly referred to as the solution space [2]. The progression of research in the domain of search has led to alternative spaces being defined. For example genetic programming (GP),[3] searches a space consisting of programs and hence this is known as a program space [3]. In the field of automated design, the space that is searched is the design space [4]. Another search space which is commonly gaining traction is the heuristic space [1, 4, 5]. Each point in a heuristic space represents a heuristic or combination of heuristics [4]. A domain barrier exists between the technique searching the heuristic space and the heuristics themselves [4, 5]. This allows the search technique to be independent of the problem domain [4, 5]. A small move in the heuristic space can correspond to a large move in the solution space. The research presented in this thesis investigates searching more than one space to solve the problem at hand.

1.1 PURPOSE OF THE STUDY

Traditionally search algorithms explore a single search space. In this study, we propose a form of concurrent search of two spaces, herein termed Concurrent Bi-space Search (CBS). The CBS approach involves searching a heuristic and a solution space concurrently. Previous studies that have explored searching two spaces used a sequential approach and an interleaving approach [6–8]. The sequential approach involved a complete search of a heuristic space, followed by a complete search of a solution space [6, 7]. The interleaving approach involved alternating the search between a heuristic space and a solution space[6, 7]. A third approach was proposed by Beckedahl and Pillay [7] in which a genetic algorithm (GA) was used to optimise when the search was moved between spaces. These approaches only search in a single space at any given time. The CBS approach presented in this study differs in that

it searches the two spaces simultaneously (concurrently). In this study, the hypothesis is that searching two spaces concurrently may be more effective than searching a single space. The effectiveness of the CBS approach is tested on the one-, two- and three-dimensional bin packing problems (1BPP, 2BPP and 3BPP). The 1BPP is used to investigate the effectiveness of the approach, while the 2BPP and 3BPP are used to investigate the scalability of the approach.

1.2 AIM AND OBJECTIVES

The aim of this study is to test the hypothesis that concurrently searching the heuristic and solution spaces, i.e. a concurrent bi-space search, CBS, is more effective than searching a single space.

1.2.1 Objectives

The objectives to achieve this aim are:

1. To design and develop an algorithm that searches the solution space to solve bin packing problems.
2. To design and develop an algorithm that searches the heuristic space to solve bin packing problems.
3. To design and develop a CBS algorithm that searches the heuristic and solution spaces concurrently.
4. To compare the effectiveness of CBS to that of solution space search.
5. To compare the effectiveness of CBS to that of heuristic space search.

1.3 SCOPE OF THE STUDY

The scope of this study is as follows:

- **Local Search (LS):**
  The CBS approach will be developed using a local search. Both the solution and heuristic space searches will also use a local search. Local search was chosen over more complex optimisation techniques, such as evolutionary algorithms, due to its lower computational cost.
- **Search Spaces:**
  The CBS approach will concurrently search a heuristic and a solution space. The effectiveness of

the CBS approach will be compared to that of solution space search and heuristic space search. The same low-level, i.e. application-specific heuristics, used by the heuristic space search will be used by the CBS approach.

- **Bin Packing Problems:**
  The effectiveness of the CBS approach compared to that of the single space searches will be evaluated using bin packing problems. The 1BPP was chosen to investigate the effectiveness of the CBS approach, whilst the 2BPP and 3BPP will be used to evaluate the scalability of the proposed approach.
- The evaluation of the performance of the CBS approach will be assessed using the number of bins and the fitness value, which is a measure of the total wasted space across the bins. The computational times of the searches will also be compared.

1.4 CONTRIBUTIONS

The main contribution of this work is a novel approach to bi-space search in which a heuristic space and a solution space are explored simultaneously for solving bin packing problems. In previous studies on bi-space search, only a single space is explored at any given time. Traditionally search is conducted in a single space using uniform information contained in that search space to navigate the search algorithm toward an optimal state [2]. In the proposed approach information from two search spaces is concurrently used to drive the algorithm to the goal state.

1.5 OUTLINE OF THE THESIS

The remainder of this thesis is structured as follows:

1.5.1 Chapter 2: Related Work

A brief overview of combinatorial optimisation problems and search techniques is presented in Chapter 2. The chapter begins with a general overview of combinatorial optimisation problems and the search process. This is followed by a general discussion of different types of search techniques, specifically the search spaces that they explore. This includes hyper-heuristics. The chapter leads on to a discussion of bi-space search techniques and concludes with an overview of packing problems.

1.5.2 Chapter 3: Bin Packing Problems

Chapter 3 presents a review of selected studies focused on solving 1BPP, 2BPP and 3BPP. The studies presented are those that have had a significant impact on the field of solving bin packing problems or have produced state-of-the-art results. Both studies that present techniques which explore a solution space and those that explore a heuristic space, i.e. hyper-heuristics, are discussed.

1.5.3 Chapter 4: Research Methodology

This chapter begins with a presentation and discussion of the research methodology used in this study. The chapter then presents the benchmark datasets for each of the three problem domains that are used in this study, followed by a description of the performance measures that will be used to evaluate the effectiveness of the CBS approach and the statistical tests that will be used to determine the significance of the results obtained. The chapter concludes with a description of the technical specifications for the simulations.

1.5.4 Chapter 5: Concurrent Bi-Space Search

Chapter 5 presents the CBS approach. The chapter begins with a general overview of the local search technique that is used to guide the single space searches against which the effectiveness of CBS is compared as well as the CBS approach itself, discussing the general structure and components common to all three approaches. The chapter proceeds with a discussion of the details of the local search that are specific to each of the three approaches, namely SSS, HSS and CBS.

1.5.5 Chapter 6: Results and Discussion

Chapter 6 presents the results and a discussion of the experiments conducted to investigate the effectiveness of the CBS approach. The chapter begins with a comparison of the performance of the CBS approach to that of the single-space approaches, SSS and HSS. An in-depth analysis of the CBS approach is then presented, including the effect of different move acceptance approaches, followed by a comparison of the CBS approach to previous bi-space search approaches. The chapter concludes with a comparison of the CBS approach to state-of-the-art approaches.

1.5.6 Chapter 7: Conclusions and Future Work

Finally, Chapter 7 presents the conclusions of this study, as well as suggestions for future work.

CHAPTER 2 RELATED WORK

This chapter discusses combinatorial optimisation problems and search techniques, and provides a general overview of packing problems. The chapter begins with an overview of combinatorial optimisation problems (Section 2.1). This is followed by a discussion of solution space search (Section 2.2), hyper-heuristics (Section 2.3) and bi-space search (Section 2.4). The chapter concludes with a discussion of packing problems (Section 2.5).

2.1 COMBINATORIAL OPTIMISATION PROBLEMS

Combinatorial optimisation problems are discrete optimisation problems in which the set of possible solutions is finite. These problems can be classified into two main classes: P and NP [9]. For those classed as P, there exists an algorithm that can solve them in polynomial time [9]. The class NP (non-deterministic polynomial time) contains problems for which there exists an algorithm that can be verified in polynomial time [1]. Some problems are known as being either NP-hard or NP-complete. For both classifications, there is no guarantee that an algorithm exists which can solve them in polynomial time. However, an NP-complete problem can be reduced to another problem which can be verified in polynomial time, whereas an NP-hard problem cannot.

The process of solving combinatorial optimisation problems involves some manner of searching through the finite set of solutions to find an optimal element [1, 2]. The set of elements searched is commonly referred to as the search space [2]. More formally, the search space can be defined as a set of states, $S = \{s_1, \ldots, si\}$, and associated operators, $O = \{o_1, \ldots, o_j\}$, that can be applied to these states, such that $o_k(s_l) \rightarrow s_m$ [1, 2]. In other words the operators are functions that enable the search to move from one state in the space to another.

2.2 SOLUTION SPACE SEARCH

In cases where these states are solutions to a problem, the search space is commonly referred to as a solution space. For example, in the bin packing problem (BPP), a solution/state might be a list of bins and a list of items placed in those bins, whilst an operator might be to place an item in a bin. The process of searching through a solution space is referred to as solution space search [2]. Exact methods are those that guarantee to find the optimal solution to a problem [2]. These methods perform an exhaustive, or near-exhaustive, exploration of the solution space, i.e. checking each to see if it is the optimum. For this reason, they tend to have high computational costs and quickly become impractical for solving NP-hard problems with large search spaces.

Heuristic methods were proposed as a means of overcoming the limitations of exact methods [2]. These methods do not guarantee finding the optimal solution, but rather find an acceptable solution in a reasonable amount of time [1, 2]. They typically achieve this by using some form of heuristic information, derived from the problem domain, to guide the search process towards an effective solution [1, 2]. This heuristic information is used to guide the search process in several ways, for example by guiding the search towards promising regions of the search space or by guiding the search towards solutions that are similar to those that have been found to be effective in the past [1, 2].

2.3 HYPER-HEURISTICS

A natural extension to heuristic methods, and one that has gained traction in recent years, is the concept of hyper-heuristics [5, 6]. Hyper-heuristics are search methods that explore a search space of heuristics, or *heuristic* space, rather than a solution space [4]. The heuristic space contains heuristics or combinations of heuristics. These domain-specific heuristics are sometimes referred to as low-level heuristics (LLHs) [4]. Hyper-heuristics can be loosely thought of as '*heuristics to choose heuristics*' [5]. More formally, hyper-heuristics have been defined as a search technique or learning mechanism which is used to select existing heuristics or generate new ones [4, 5]. From this definition hyper-heuristics can be classified as either selection hyper-heuristics or generation hyper-heuristics [4, 5]. Selection hyper-heuristics are those that select heuristics from a predefined set of heuristics [4, 5], while generation hyper-heuristics are those that generate new heuristics [4, 5]. Hyper-heuristics are further classified as either constructive [4, 5], which build a solution from scratch, or perturbative [4, 5], which modify an existing solution.

Hyper-heuristics function by finding an effective method of solving a given problem [4], in the form of one or more LLHs, rather than directly finding a solution to the problem [4]. This is achieved by searching a space of LLHs (i.e. a heuristic space), rather than the traditional solution space [4, 5]. The effectiveness of a hyper-heuristic is dependent on the quality of the LLHs [4], or heuristic components in the case of generative hyper-heuristics, that are available to it. Typically, the LLHs incorporate knowledge from the problem domain [4], whilst the higher level hyper-heuristic is usually problem independent, operating only on a heuristic space.

Given that each LLH operates on or creates a solution, it follows that there exists a mapping between points in the heuristic space and points in the solution space. This mapping is frequently a many-to-one mapping from the heuristic space to the solution space. An example of this can be seen with bin packing where the construction LLHs *first fit* and *worst fit* [10] would both place the first item in the same bin, leading to the same point in solution space. Through this mapping between the two search spaces, hyper-heuristics can explore a broader area of the solution space than exploring the solution space directly [4] as small changes in the heuristic space can correspond to large changes in the solution space [4]. It is to be noted that while each point in the heuristic space can be mapped to a corresponding point in the solution space, the reverse is not necessarily true [4].

2.4 BI-SPACE SEARCH

Bi-space search aims to explore two search spaces, thereby maximizing the advantages and minimizing the disadvantages that are inherent in each space.

To the best of the author's knowledge, the first to investigate the effectiveness of searching across more than one space was Qu and Burke [6]. In the study the authors presented a graph-based hyper-heuristic (GHH) for solving the course and exam timetabling problems. The study also proposed two approaches for hybridising search between the heuristic and solution spaces, termed GHH1 and GHH2. GHH1 involved local improvement on complete solutions by searching the heuristic space to find a heuristic sequence which was used to build a complete solution. A greedy search was then performed on this complete solution. GHH2 involved local improvement during solution construction. This was achieved as follows. During the evaluation of a heuristic sequence, the first LLH in the sequence is applied to obtain a partial solution. A greedy search is then performed on this partial solution until there is no improvement at which point the next heuristic in the sequence is applied to obtain a new partial

solution. The greedy search is again applied to this new partial solution. The process is repeated until the entire heuristic sequence has been evaluated, and a final greedy search is applied. For the course timetabling problem, both GHH1 and GHH2 were found to outperform the GHH approach for all instances except the large instance where GHH1 performed slightly worse and GHH2 had equal performance. GHH2 was able to outperform the state-of-the-art approaches for all the small instances and two of the medium instances. For the exam timetabling problem, GHH1 had equal performance to GHH for all instances except the *hec92* and *uta93* instances for which it performed better and worse respectively. GHH2 outperformed GHH for all instances except the *yor83* instance for which it performed worse. Neither GHH1 nor GHH2 were able to compete with the state-of-the-art for exam timetabling. For both the course and exam timetabling problems GHH2 outperformed GHH1. There are two main drawbacks to the GHH1 and GHH2 approaches. The first is that the heuristic space is always the first to be searched when it may be more optimal to begin with a solution space search. The second drawback is that the switch between search spaces occurs at fixed points in the search process. In GHH1 the switch occurs after the entire heuristic sequence has been applied whilst in GHH2 the switch between spaces is forced to occur after the application of each heuristic. Neither of these cases allow for a scenario in which it is more optimal to apply more than one heuristic before switching spaces. GHH2 has an additional drawback in that it may be more beneficial to perform a partial search in the solution space before returning to the heuristic space, as opposed to the complete search.

Beckedahl and Pillay [7] later applied the concepts of GHH1 and GHH2 to 1BPP, labelling their version of GHH1 the sequential search approach (SSA) and GHH2 the interleaving search approach (ISA). To overcome the drawbacks discussed above, the authors introduced a third approach which they termed the concurrent search approach (CSA). In the study all three search approaches were driven by a genetic algorithm hyper-heuristic (GAHH) which searched through sequences of constructive LLHs. For both SSA and ISA the evaluation of an LLH sequence in the GAHH was performed as described for GHH1 and GHH2 respectively. For the CSA the local search move operator was included as a possible LLH in the heuristic sequences being searched by the GAHH. During the evaluation of a sequence, each time the local search move operator was encountered it would be applied only once (as opposed to repeatedly until no improvement). All three bi-space approaches were found to outperform the standard GAHH, with the CSA being the best performing, solving 1107 out 1210 problem instances (the *Scholl* benchmark datasets discussed in Section 4.2.1) to optimality and a further 88 instances to within one bin of the optimum. Whilst CSA was found to perform best among the four approaches, it was reported to be not competitive with the state-of-the-art. In addition, the approach had a high

computational cost.

To reduce the computational cost, Beckedahl and Pillay [8] later extended their work, proposing an approach the authors termed bi-space search (BSS). The approach used local search (LS) to optimise the switching between the heuristic (HSS) and solution (SSS) space searches. Both HSS and SSS also employed LS. The three searches were evaluated and compared against one another using the 1BPP. BSS was found to outperform both HSS and SSS for all 1BPP benchmark datasets tested (all datasets discussed in Section 4.2.1), finding the optimum solution for 1336 out of 1615 instances and solutions of one bin from the optimum for a further 164 instances. Although BSS found better quality solutions than both HSS and SSS, it was found to have longer runtimes than the single-space searches. BSS was marginally outperformed by CSA with a difference of 23 instances solved to optimality between the two approaches. As with the previous approaches to bi-space search, BSS produced better quality solutions than the single-space searches, but the single-space searches had a lower computational cost. This study aims to reduce the runtimes of bi-space search by presenting the CBS approach which searches the two spaces simultaneously.

2.5 PACKING PROBLEMS

Cutting and packing problems are a class of combinatorial optimisation problems that cover a variety of problems with many relevant applications to industry and logistics [11], and are known to be NP-hard [10, 11]. Some examples of cutting and packing problems are as follows:

- knapsack problems
- bin packing problems
- cutting stock problems
- strip packing problems

The knapsack problem (KP) can be formally defined as follows. Given a list of n items each with an associated cost c_i (multiple items can have the same cost) and a unique volume v_i (unique dimensions) and a knapsack (bin) of set capacity b. The task is to find a configuration (packing) of the items (allowing the same item to occur multiple times) that fits within the knapsack (i.e., does not exceed its capacity b) such that the total cost of the items in the knapsack is a maximum [10].

The bin packing problem (BPP) is similar to the knapsack problem. Given a list of items each having an associated size (different items are allowed to have the same size) and an infinite number of bins of fixed capacity c, place all items into a bin such that no bin exceeds its capacity, no items overlap one another within a bin and the total number of bins used is a minimum [10]. There are also variants of the BPP in which the sizes of the bins are variable.

The cutting stock problem (CSP) is identical to the BPP with the exception that each item has an associated demand specifying the required number of occurrences of said item in the final packing [10]. The KP, BPP and CSP all have one, two and three-dimensional variants. The three-dimensional BPP is also referred to as the container loading problem (CLP).

The strip packing problem (SPP) is a two or three-dimensional problem which involves packing a list of items into a single bin where one of the dimensions is set to infinite size and the remaining dimensions are fixed. The objective is to pack all items into the bin in such a way that the space used in the infinite dimension is a minimum [10]. The two-dimensional SPP fixes the width of the bin and the height is infinite whilst for the three-dimensional case both the width and height are fixed, and the length is infinite [10].

For each of the two and three-dimensional cutting and packing problems mentioned above there exist variants in which item rotations are allowed as well as the variations in which the *guillotine cuts* constraint is applied. The guillotine cuts constraint requires that the items be packed in a way such that the items can be obtained through a series of cuts made between parallel edges of either the bin or a piece resulting from a previously made (guillotine) cut [10].

The BPP can be formally defined as follows. Given a list of items, each having a specific length (1D), area (2D) or volume (3D), denoted w_i, and an infinite number of containers of fixed size C, each item is to be placed into a container such that no item exceeds the bounds of its container (i.e. the capacity is not exceeded), no two items within a given container overlap, and the total number of containers used is a minimum.

Dyckhoff [12], Lodi et al. [13] and Wäscher et al. [14] each proposed a different typology for classifying cutting and packing problems. The Dyckhoff typology [12] classifies cutting and packing problems according to four different criteria as follows:

1. Dimensionality of the problem:

   (1) one-dimensional

   (2) two-dimensional

   (3) three-dimensional

   (N) N-dimensional with $N > 3$

2. Type of assignment:

   (B) assign a selection of items to all objects/containers

   (V) assign all items to a selection of objects/containers

3. Variety of large objects/containers:

   (O) one object/bin/container

   (I) same size objects/containers

   (D) different size objects/containers

4. Variety of small items:

   (F) few items (of different sizes)

   (M) many items (of many different sizes)

   (R) many items with relatively little variation (i.e. non-congruent) in their sizes

   (C) same size items (congruent)

Lodi et al. [13] proposed a typology for two-dimensional packing problems that considers whether the items are allowed to be rotated and whether the guillotine cuts constraint is applied. The typology is as follows.

- 2BP|O|G: two-dimensional bin packing problem, items cannot be rotated (i.e. are oriented) and the guillotine cuts constraint is applied
- 2BP|R|G: two-dimensional bin packing problem, items can be rotated and the guillotine cuts constraint is applied

- 2BP|O|F: two-dimensional bin packing problem, items cannot be rotated (i.e. are oriented) and the cutting is free (no guillotine constraints)
- 2BP|R|F: two-dimensional bin packing problem, items can be rotated, and the cutting is free (no guillotine constraints)

The typology proposed by Wäscher et al. [14] was developed taking into consideration the drawbacks experienced by the previous two typologies. The proposed typology uses five different criteria to classify cutting and packing problems, namely dimensionality, type of assignment, variety of large objects/containers, variety of small items and shape of the small items.

CHAPTER 3 BIN PACKING PROBLEMS

This chapter provides a review of the literature focused on solving BPPs. The studies presented are those that solved BPPs by exploring the solution space, as well as those that explored the heuristic space. First, studies which solved 1BPP are discussed (Section 3.1), followed by those which solved 2BPP (Section 3.2) and 3BPP (Section 3.3).

All the BPP variants discussed in Section 2.5 can be classed as either *online*, in which the items are presented one at a time and must be packed immediately [10, 15], or *offline*, in which all items are known in advance and can be packed at the same time [10, 15]. For the higher dimensional BPPs, there exist two additional classifications, namely *regular* and *irregular* [16]. In the regular case, the items are rectangular, whereas in the irregular case the items are polygons [16]. Using the typologies discussed in Section 2.5, the BPPs used in this study are classified as 1/2/3|V|I|M (Dyckhoff [12]), 1/2/3BP|O|F (Lodi et al. [13]) or 1/2/3SBSBPP (Wäscher et al. [14]), for the *offline*, *regular* case. This choice was made as these are the most commonly used BPPs in the literature, allowing for comparison with a broader range of studies. The studies reviewed in the sections that follow involved solving the offline, regular BPPs.

3.1 ONE-DIMENSIONAL BIN PACKING PROBLEM

This section provides an overview of studies that have had an impact on the field of solving the 1BPP as well as studies that have produced state-of-the-art techniques for the 1BPP. Techniques which explore the solution space are discussed first, followed by those that explore the heuristic space. The nine benchmark datasets for 1BPP that are commonly mentioned in the studies below are discussed in Section 4.2.1.

3.1.1 Solution Space Search

The following studies solved the 1BPP by exploring a solution space:

Martello and Toth [17] proposed a reduction procedure (MTRP) for exactly solving the 1BPP, which iteratively reduces the problem size. The authors define a dominance criterion, in which a bin *A* dominates bin *B* if bin *A* has the same amount of space or more than bin *B* but contains fewer items. The authors also defined lower bounds. At each iteration in the MTRP procedure, the lower bounds and the dominance criterion are used to find the best set of items to place into a bin. This set of items is then removed from the problem instance to create a reduced problem, to which the next iteration is applied. The process is repeated until all items have been packed. The authors generated their 1BPP problem instances for the study.

Scholl et al. [18] proposed a hybrid procedure for exactly solving the 1BPP. The proposed approach, which was termed BISON (**bi**n packing **so**lutio**n** procedure), used new lower bounds and the dominance criterion defined by Martello and Toth [17] in the study above to reduce the problem size. The solution to the reduced problem was then refined using a tabu search [19]. BISON was found to outperform the state-of-the-art at the time. The authors also introduced the commonly used *Scholl* benchmark datasets, *Scholl_1*, *Scholl_2* and *Scholl_3*.

Falkenauer [20] proposed a technique for the 1BPP which hybridised the dominance criterion, as defined by Martello and Toth [17], with a grouping genetic algorithm (GGA) [21]. The dominance criterion was used both to enhance the effectiveness of the genetic operators and to refine the offspring produced by said operators. The proposed hybrid GGA performed better than a standard GGA and the MTRP. The authors also presented the *Falkenauer* uniform and triplet benchmark datasets.

Alvim et al. [22] proposed a hybrid improvement procedure (HI-BP) for 1BPP which begins with a pre-processing step that reduces the size of the problem by forcibly packing some items into certain bins, creating a reduced problem. An initial candidate solution is then created for the reduced problem using a greedy construction algorithm and then improved using a tabu search [19] augmented with a load redistribution technique. The approach was tested on the *Falkenauer*, *Scholl*, *Schwerin* and *Hard28* datasets. The proposed HI-BP approach found the optimum number of bins for 1582 problem instances out of the total 1587.

Fleszar and Hindi [23] proposed a variable neighbourhood search (VNS) technique for 1BPP which used modified versions of the minimum bin slack (MBS) heuristic [24] to create an initial solution, which was then improved through the VNS. The proposed approach was tested on the *Scholl* and *Falkenauer* datasets and found the optimum solution for 1329 instances out of the total 1370.

Fleszar and Charalambous [25] adapted the techniques presented in Fleszar and Hindi [23] by introducing a sufficient average weight (SAW). The authors defined a subset of items as having a sufficient average weight if the average weight of all items in the subset was at least as large as the average weight of all items still to be packed, including the items in the subset. The perturbation SAWMBS technique (Pert-SAWMBS) was able to find the optimum number of bins for 1590 problem instances out of 1615 from the nine benchmark datasets on which it was tested. The Pert-SAWMBS technique is one of the state-of-the-art approaches against which the CBS approach is compared in Section 6.5

Quiroz-Castellanos et al. [26] used a grouping genetic algorithm [21] with controlled gene transmission (GGA-CGT) to solve the 1BPP. The authors defined new genetic operators which operate at the bin level. The GGA-CGT was tested on all nine 1BPP benchmark datasets, finding the optimum number of bins for 1602 problem instances out of the total 1615 on which it was tested. The GGA-CGT is one of the state-of-the-art techniques against which the CBS approach is compared in Section 6.5.

Buljubašić and Vasquez [27] presented a consistent neighbourhood search technique (CNS-BP) for solving 1BPP and 2D vector packing. The technique randomly sorts the items and applies a simple reduction procedure followed by the first fit construction heuristic [28] to create an initial solution. A tabu search [19] is then used in conjunction with local moves to iteratively reduce the number of bins. The technique was applied to all nine benchmark datasets, finding the minimum number of bins for 1612 out of the total 1615 problem instances. CNS-BP is another state-of-the-art technique against which CBS is compared in Section 6.5.

3.1.2 Heuristic Space Search

This section provides an overview of studies using hyper-heuristics to solve the 1BPP.

An evolutionary algorithm (EA) [29] selection hyper-heuristic was proposed by Ross et al. [30] for solving 1BPP. The proposed approach uses an XCS learning classifier system [31] to learn

a set of rules to associate different characteristics of the problem (which arise at different points during the solution process) to different LLHs. The classifier system was tested on the *Falkenauer_T*, *Falkenauer_U*, *Scholl_1* and *Scholl_3* datasets, and found the optimal solution for 78.1% of the instances, outperforming each of the LLHs applied independently. The authors do not report a breakdown of the number of instances for which the optimum solution was found for each of the datasets.

Ross et al. [32] later proposed a messy-genetic algorithm (messy-GA) based selection hyper-heuristic to learn a combination of LLHs for solving 1BPP. The authors tested their messy GA on the same 890 benchmark problem instances as in Ross et al. [30] and an additional randomly generated 126 instances, totalling 1016 instances. The set of instances was split into training and testing sets consisting of 763 and 253 instances respectively. The messy GA performed better than each of the LLHs applied independently for both training and testing sets, as well as outperform the XCS classifier system [31]. The messy GA found the optimum solution or better for 98.3% of the training instances and 97.6% of the testing instances. The authors do not report a breakdown of the results for the respective datasets.

Sim et al. [33] proposed a selection hyper-heuristic classifier for 1BPP. The proposed approach used an evolutionary algorithm (EA) [29] to evolve predictor attributes for a k-nearest neighbour classifier system, which was used to select the best LLH for an unseen problem instance. The approach was evaluated using the 1370 problem instances from the *Falkenauer* and *Scholl* datasets. The hyper-heuristic found the optimum solution for 521 (76.06%) problem instances. The authors do not report a breakdown of the results for the respective datasets.

Burke et al. [34] proposed a genetic programming (GP)-based [3] generative hyper-heuristic for the online 1BPP. The proposed GP system evolved LLHs that emulated the functionality of the well known, human designed, first-fit heuristic [28]. This was achieved by evolving programs which took into account the size of the item to be packed, the capacity of the bin and the fullness of the bin. The approach was tested on 20 instances taken from the *Falkenauer* datasets. The evolved heuristics were found to perform as well as the first-fit heuristic for all 20 instances.

A simulated annealing [35] selection hyper-heuristic (SAHH) was proposed by Bai et al. [36] for solving the course timetabling problem and 1BPP. The SAHH employed a stochastic selection strategy

in conjunction with a short-term memory and maintained a domain barrier between the pool of LLHs and the higher-level search. For 1BPP the SAHH was tested using the *Falkenauer*, *Scholl*, *Schwerin* and *Wäscher* datasets. When averaged across 20 runs per problem instance, the SAHH found the optimum solution for 1561.9 instances out of the total 1587, being outperformed by the state-of-the-art at the time. The SAHH approach is one of the state-of-the-art techniques against which the CBS approach is compared in Section 6.5.

Sim and Hart [37] presented a hyper-heuristic for 1BPP which used a single node GP (SNGP) [38] island model. The SNGP is used on its own as a generative hyper-heuristic to evolve new deterministic LLHs for 1BPP. The authors also incorporated SNGP into an island model as a selection hyper-heuristic to evolve sets of LLHs. The two approaches were tested on the 1370 problem instances from the *Scholl* and *Falkenauer* datasets, split equally into training and testing sets. The best heuristic generated by the SNGP found the optimum solutions for 518 instances, outperforming the human-derived heuristics. The island model which combined LLHs found the optimum for 559 problem instances when using the generated heuristics, outperforming the island model which used only human-derived heuristics.

Burke et al. [39] proposed a generative hyper-heuristic using grammatical evolution (GE) [40] to generate new perturbative LLHs for solving 1BPP. The GE system was tested using the following sets of benchmark instances: *Falkenauer_U500*, *Falkenauer_U1000*, *Falkenauer_T501* and *Scholl_3*. The GE system was run separately for each set of instances, using the first instance for training and the remaining instances for testing the evolved heuristics. The evolved heuristics were found to perform well, on average finding solutions that were optimal or within one bin of the optimum.

López-Camacho et al. [41] present a GA-based [29] selection hyper-heuristic for solving 1BPP and 2BPP. In the case of 2BPP the authors considered the regular case (rectangular items) as well as two other variants in which the items are convex and non-convex polygons. The proposed approach was applied to 397 1BPP problem instances taken from the *Falkenauer*, *Scholl* and *Wäscher* datasets. The authors do not specify which instances were taken from which dataset. The 540 2BPP instances were taken from those generated in Terashima-Marín et al. [16]. The proposed hyper-heuristic on average found better solutions than the single heuristics for 1BPP, but worse solutions for 2BPP.

Burke et al. [42] proposed a GP-based [3] generative hyper-heuristic (GPHH) for solving 1D, 2D and 3D knapsack and bin packing problems. The proposed approach was evaluated using 18 different

datasets across the different problem domains. For 1BPP the *Falkenauer_U* and *Scholl_3* datasets were used. The authors used the following 2BPP datasets in the study: *beng* [43], *ngcut* [44], *gcut* [45] and *cgcut* [46]. The *thpack9* [47] dataset was used for 3BPP. For 1BPP the GP system had mixed results when compared with state-of-the-art. The GPHH had comparable performance to state-of-the-art for the *Falkenauer_U* dataset, but had worse performance for the *Scholl_3* dataset. For 2BPP the proposed approach had comparable performance on average across the four datasets, however for 3BPP the performance was worse than the state-of-the-art.

3.2 TWO-DIMENSIONAL BIN PACKING PROBLEM

This section provides an overview of studies which have had an impact on solving the 2BPP as well as studies that have produced state-of-the-art techniques for the 2BPP. Techniques which explore the solution space are discussed first, followed by those that explore the heuristic space.

3.2.1 Solution Space Search

Martello and Vigo [48] proposed an exact technique for solving 2BPP based on the branch-and-bound algorithm [49]. The approach used a 2D version of the MTRP [17] to reduce the problem size. The reduced problem is then solved using new upper and lower bounds that the authors presented. The authors applied their approach to the following 2BPP datasets: *beng* [43], *ngcut* [44], *gcut* [45] and *cgcut* [46]. The authors also generated their own 2BPP problem instances to augment the dataset presented by Berkey and Wang [50]. These combined 500 problem instances form the commonly used *class* dataset for 2BPP. Given a time limit of 300 seconds, the approach was able to find the optimum solution for 484 problem instances out of the total 500 for the *class* dataset.

Lodi et al. [51] presented a tabu search [19] approach for solving 2BPP (TS2). The authors create an initial solution for the tabu search by treating the problem first as a strip packing problem (packing the items into a strip of infinite height and fixed width). Once the items have been packed into the strip, the strip is then cut into pieces and the problem is solved as a 1BPP, treating each piece as an item. The tabu search then improves this solution using one of two moves, namely to directly move an item from one bin to another or to recombine the items from across two different bins. The tabu search approach was found to outperform both the finite first fit [50] and the finite best strip [50] heuristics. The approach was applied to the *beng* [43], *ngcut* [44], *gcut* [45], *cgcut* [46] and *class* [48, 50] datasets. The TS2 approach found the optimum number of bins for 12 out of the 50 subclasses of problem

instances in the *class* dataset. The TS2 approach is one of the state-of-the-art-techniques for 2BPP against which the CBS approach is compared in Section 6.5.

Boschetti and Mingozzi [52] proposed an approximation technique for solving 2BPP which the authors termed a heuristic for bin packing (HBP). The approach involves assigning a price to each item and using some criterion to determine the order in which the items are packed. The authors presented different methods both for assigning prices to the items and for how each item is allocated to a bin. The HBP technique enumerates through all possible combinations of the different methods and selects the best combination. HBP was able to find the optimal solution for 384 out of the 500 instances on the 2BPP *class* [48, 50] dataset. The HBP approach is one of the state-of-the-art-techniques for 2BPP against which the CBS approach is compared in Section 6.5.

Monaci and Toth [53] proposed an approximation technique which the authors termed a set-covering-based heuristic (SCH) approach. SCH formulates 2BPP as a set-covering problem. The approach involves a *column generation* phase and a *column optimisation* phase. The column generation phase generates feasible item sets (columns). The column optimisation phase then solves the set-covering problem using a Lagrangian-based algorithm. SCH found the optimum solution for 430 out of the total 500 problem instances on the 2BPP *class* [48, 50] dataset. The SCH approach is one of the state-of-the-art-techniques for 2BPP against which the CBS approach is compared in Section 6.5.

Faroe et al. [54] presented a guided local search (GLS) approach for solving 2BPP and 3BPP. An initial solution is created for the GLS using a shelf approach which involves first packing the items into two-dimensional "shelves" of fixed width and length, with a height equal to that of the boxes. The shelves are then treated as items in a 1BPP and packed accordingly. This approach is similar to that used by Lodi et al. [51]. The GLS then uses local search moves and a memory of beneficial and detrimental features from previous solutions to make improvements and guide the search to more promising areas of the search space. The GLS approach was applied to the *class* [48, 50] 2BPP dataset and found the optimum solution for 194 out of the 500 instances. GLS was also applied to the 3BPP dataset by Martello et al. [55], on which it found the optimum for 16 out of 320 instances. The GLS approach is one of the state-of-the-art-techniques for 2BPP and 3BPP against which the CBS approach is compared in Section 6.5.

A greedy randomised adaptive search procedure [56] (GRASP) hybridised with a variable neighbourhood descent (VND) was proposed by Parreño et al. [57] for solving 2BPP and 3BPP. The GRASP component consists of a constructive phase, an improvement phase and a diversification phase. The constructive phase involves iteratively packing remaining items into a single bin until all items have been packed and incorporates a randomisation strategy. The authors define four possible moves for the improvement phase, each one defining a neighbourhood structure. These neighbourhood structures are then used by VND to improve the solution. If the approach is unable to find an improvement for a specified number of iterations, a modified version of the constructive phase (i.e. the diversification phase) is used on the next iteration. The diversification phase involves prioritising the packing of items which have most commonly been left unpacked in previous iterations. For the 2BPP *class* [48, 50] dataset GRASP-VND (GVND) found the optimum solution for 430 out of the 500 instances and had equal or better performance than the state-of-the-art for 48 out of the 50 problem instance classes. GVND was found to be the best or tied for best performing for 27 out of the 32 classes of 3BPP problem instances from Martello et al. [55] when compared to the state-of-the-art at the time. However, the approach was only able to find the optimum solution for 23 out of the total 320 instances. The GVND approach is one of the state-of-the-art-techniques for 2BPP and 3BPP against which the CBS approach is compared in Section 6.5.

Gonçalves and Resende [58] proposed a biased random key genetic algorithm (BRKGA) for solving 2BPP and 3BPP, both with and without item rotations. BRKGA represented a solution as a vector of random keys, with each key corresponding to an item. The keys are then sorted to determine the order in which the items are packed. For 2BPP the BRKGA was found to have equal or better performance than the state-of-the-art approaches for all 50 classes of problem instances from the *class* [48, 50] dataset and found the optimum number of bins for 26 out of the 50 classes. Across all 500 2BPP problem instances the BRKGA approach used 61 extra bins. BRKGA was applied to the 3BPP instances from Martello et al. [55] and was found to have equal or better performance than the state-of-the-art for 30 out of the 32 classes of problem instance. The authors do not report the number of 3BPP problem instances for which the optimum solution was found. The BRKGA approach is one of the state-of-the-art-techniques for 2BPP and 3BPP against which the CBS approach is compared in Section 6.5.

3.2.2 Heuristic Space Search

Terashima-Marín et al. [16] proposed a GA-based [29] selection constructive hyper-heuristic for solving the regular and irregular (convex polygonal) 2BPP. Each individual in the population of the GA specifies the LLH to be applied at each point in the solution process, i.e. at each state of the problem. For the case of the regular 2BPP the approach was evaluated using the 500 benchmark instances proposed by Berkey and Wang [50] and Martello and Vigo [48], the instances proposed by Terashima-Marín et al. [59] and a set of randomly generated instances created by the authors, all totalling 1080 instances which were divided at random into training and testing sets of 540 instances each. The hyper-heuristics were found to perform as well as or better than the single heuristics for 89.46% of instances for the training set and 85.6% of instances in the testing set.

Beyaz et al. [60] proposed a memetic algorithm (MA)-based selection hyper-heuristic for 2BPP, both with and without item rotations. The authors termed their approaches Hyper-Heuristic Algorithm-O/NO (HHA-O/HHA-NO) for the oriented and non-oriented 2BPP cases respectively. Each individual in the population of the MA specifies the order in which the items are packed as well as two LLHs to be used during packing, one for each half of the items. The approach was applied to the 500 benchmark instances proposed by Berkey and Wang [50] and Martello and Vigo [48]. The MA was able to find the optimum solution for 11 out of the 50 classes of problem instances without rotations, and 9 classes allowing item rotations. On average the proposed MA was able to find better solutions than the single heuristics, however at the cost of longer runtimes. The HHA-O approach is one of the state-of-the-art approaches against which the CBS approach is compared in Section 6.5.

Terashima-Marin et al. [61] proposed two selection hyper-heuristics for solving 2BPP. The first was a XCS-based learning classifier system [31], which was trained to learn a solution procedure when solving a given problem, and the second was a GA-based [29] approach which evolved combinations of condition-action rules. Both of the proposed approaches were evaluated using problem instances taken from Beasley [62], Berkey and Wang [50] and Martello and Vigo [48], Terashima-Marín et al. [59] and a set of instances randomly generated by the authors, all totalling 1080 problem instances. The total set of 1080 instances was divided at random into two subsets of 540 instances each. Both approaches had mixed performance, finding solutions that were one bin better or one bin worse than the single heuristics for a small percentage of the instances. The majority of the instances were solved to the same number of bins as the single heuristics.

3.3 THREE-DIMENSIONAL BIN PACKING PROBLEM

This section provides an overview of those studies which have had an impact on the field in solving the 3BPP as well as studies that have produced state-of-the-art techniques for the 3BPP. Techniques which explore the solution space are discussed first, followed by those that explore the heuristic space.

3.3.1 Solution Space Search

Lodi et al. [63] proposed an adapted version of their previous tabu search [19], in Lodi et al. [51], to solve 3BPP (TS3). The proposed TS3 employs a new construction technique which works in two phases. The first phase clusters the items into layers based on their relative heights. The items in each cluster are then sorted according to the areas of their bases. The order of the clusters together with the sub-ordering within each cluster determines the overall order in which the items are packed in the different layers. The different heights of each layer are treated as items and the problem is then solved as a 1BPP. Phase two directly sorts the items according to their base areas and initialises each of the layers produced by phase one, without packing any items into the layers. Phase two then packs and combines the layers as in phase one. The best solution found between the two phases is the one which is returned by the technique. This construction technique was used as the move operator in the tabu search. The TS3 approach was applied to classes one and four to eight of the 3BPP dataset from Martello et al. [55] and outperformed the GLS approach for 12 out of the 24 problem categories and performed worse for 11 out of 24. The authors do not report the number of instances for which the optimum solution was found. The TS3 approach is one of the state-of-the-art-techniques for 3BPP against which the CBS approach is compared in Section 6.5.

Crainic et al. [64] proposed a two-level tabu search [19] (TS2PACK) for solving 3BPP. The first level of TS2PACK reduces the number of bins while the second level improves the packing of items within a bin. An initial solution is created by applying the extreme point first fit decreasing (EP-FFD) heuristic [65]. The TS2PACK approach then iteratively removes a bin from the solution, using a defined metric to select the bin to remove, and iteratively repacks the items into the other bins, whilst relaxing the height constraints of the bins. If the new solution is feasible it is accepted and the bin is removed, otherwise the first level tabu search is applied. The first level tabu search focuses on optimising the total number of bins used. The second level tabu search focuses on optimising the packing of items within a bin and reducing its infeasibility. TS2PACK was applied to classes one and four to eight from the dataset in [55] where it outperformed the state-of-the-art at the time for six out of the 24 categories

of problem instances and was tied with state-of-the-art for a further 15 categories. The authors do not report the number of instances for which the optimum solution/lower bound value was found. The TS2PACK approach is one of the state-of-the-art-techniques for 3BPP against which the CBS approach is compared in Section 6.5.

CHAPTER 4 RESEARCH METHODOLOGY

This chapter describes the research methodology used in this study as well as the problem domain datasets that will be used and how the performance of the approaches will be assessed. First the research methodology used in this study is outlined in Section 4.1, followed by the benchmark datasets that will be used for each problem domain (Section 4.2). The metrics and statistical tests that will be used to assess the performance of the approaches are discussed in Section 4.3. The chapter concludes with a description of the technical specifications of the hardware and software used to run the experiments (Section 4.4).

4.1 RESEARCH METHODOLOGY

This section provides an overview of the research methodology used in this study. Johnson [66] presents the following methodologies for conducting research in the field of Computer Science:

- Proof by demonstration
- Empiricism
- Mathematical proof
- Hermeneutics

Oates et al. [67] asserted that some research methodologies are better suited than others to a given research problem. The authors also assert that it is possible for a research problem to have several suitable methodologies. The primary objective of this study is to determine whether a concurrent search of two search spaces, specifically the heuristic and solution spaces, is more effective than searching either of the spaces separately. To investigate this, an approach needs to be developed to perform optimization concurrently in both search spaces and iteratively refined to improve performance.

Hence, the *proof by demonstration* methodology is deemed to be the most appropriate choice for this study.

The '*proof by demonstration method*' involves creating an initial artefact, such as a model or algorithm [66]. This initial artefact is then iteratively refined until the desired solution is obtained, or no further improvements can be made [66]. The refinement process consists of three steps: *testing/evaluation*, *analysis* and *refinement* [66]. If there is no improvement during the testing step, the analysis step is used to determine the reasons for the lack of improvement [66]. The refinement step adjusts the artefact for the next iteration, addressing the reasons for the lack of improvement [66].

4.1.1 Proof by Demonstration Methodology

As previously stated in Chapter 1, the objective of this study is to use simple search techniques with low computational times to conduct search across the heuristic and solution spaces as opposed to searching the spaces separately, under the hypothesis that searching the spaces concurrently will lead to an improvement in performance. Under the proof by demonstration methodology, an initial algorithm is created to search the two spaces concurrently by simultaneously performing perturbations in each space. The specifics of this algorithm, as well as those of the single-space searches, are outlined in Chapter 5.

The algorithm is evaluated using sets of known benchmark datasets for 1BPP, 2BPP, and 3BPP problem domains. These datasets are selected for being the most commonly used within their respective problem domains, as well as for being used by the state-of-the-art techniques against which the CBS approach is compared to. These datasets are discussed in Section 4.2. The performance of the CBS approach is compared to that of the HSS and SSS single-space searches to determine whether there is an improvement. The performance metrics that are used, as well as the statistical tests that are used to determine the significance of the results, are discussed in Sections 4.3.1 and 4.3.2 respectively. If CBS does not outperform the single-space searches, the algorithm is refined by adjusting the parameter values and using different perturbation operators. Once the algorithm has been refined, it is re-evaluated and its performance compared to the single-space searches. This process is repeated until either CBS outperforms SSS and HSS, or no further improvements can be made. In the case of no further improvement being made, an analysis will be conducted to determine the reasons for this.

4.2 PROBLEM DOMAIN DATASETS

This section describes the benchmark datasets for 1BPP, 2BPP and 3BPP that are used to evaluate the CBS approach. The datasets were selected because they are the most commonly used in the literature, and they have been used by the state-of-the-art techniques against which CBS is being compared.

4.2.1 One-Dimensional Bin Packing Problem

A total of nine different existing benchmark datasets, totalling 1615 instances, were used to evaluate the approaches on the 1BPP. The characteristics of each dataset, except for the Scholl_2 dataset, are summarized in Table 4.1.

Table 4.1. 1BPP benchmark datasets.

Source	Name	Size	c	n	w_i
Falkenauer [20]	Falkenauer_T	80	{1000}	{60, 120, 249, 501}	[250, 500]
	Falkenauer_U	80	{150}	{120, 250, 500, 1000}	[20, 100]
Scholl et al. [18]	Scholl_1	720	{100, 120, 150}	{50, 100, 200, 500}	[1, 100]
			{100, 120, 150}	{50, 100, 200, 500}	[20, 100]
			{100, 120, 150}	{50, 100, 200, 500}	[30, 100]
	Scholl_2	480	-	-	-
	Scholl_3	10	{100000}	{200}	[20000,35000]
Schwerin and Wäscher [68]	Schwerin_1	100	{1000}	{100}	[150, 200]
	Schwerin_2	100	{1000}	{120}	[150, 200]
Schoenfield [69]	Hard28	28	{1000}	{160, 180, 200}	[1, 800]
Wäscher and Gau [70]	Waescher	17	{10000}	[57, 239]	[1, 7500]

Problem instances for the Scholl_2 dataset have the following characteristics:

- $n \in \{50, 100, 200, 500\}$
- $c \in \{1000\}$
- $\overline{w_i} \in \left\{\frac{c}{3}, \frac{c}{5}, \frac{c}{7}, \frac{c}{9}\right\}$
- $\delta \in \{20\%, 50\%, 90\%\}$

where n is the number of items, $\overline{w_i}$ is the target average weight of the items and δ is the maximum percentage that an item's weight can deviate from $\overline{w_i}$.

4.2.2 Two-Dimensional Bin Packing Problem

For the 2BPP, the 500 problem instances presented by Berkey and Wang [50] and Martello and Vigo [48] were used*. The instances are divided into ten different classes, further divided into sets of ten instances. The characteristics for each class of problem instance are summarised in Table 4.2, where W and H are the width and height of each bin, and w_i and h_i are the width and height of the i^{th} item.

Table 4.2. 2BPP problem instance characteristics.

Class	W	H	w_i	h_i	Class	W	H	w_i	h_i
1	10	10	$[1,10]$	$[1,10]$	6	300	300	$[1,100]$	$[1,100]$
2	30	30	$[1,10]$	$[1,10]$	7	100	100	$[\frac{2}{3}W,W]$	$[1,\frac{1}{2}H]$
3	40	40	$[1,35]$	$[1,35]$	8	100	100	$[1,\frac{1}{2}W]$	$[\frac{2}{3}H,H]$
4	100	100	$[1,35]$	$[1,35]$	9	100	100	$[\frac{1}{2}W,W]$	$[\frac{1}{2}H,H]$
5	100	100	$[1,100]$	$[1,100]$	10	100	100	$[1,\frac{1}{2}W]$	$[1,\frac{1}{2}H]$

4.2.3 Three-Dimensional Bin Packing Problem

The benchmark problem instances presented by Martello et al. [55]† were used for the 3BPP. The dataset consists of eight different classes, further subdivided into sets of ten instances. Table 4.3 summarises the characteristics for each class of 3BPP problem instance, where W, H and D are the width, height and depth of each bin, and w_i, h_i and d_i are the width, height and depth of the i^{th} item.

Table 4.3. 3BPP problem instance characteristics.

Class	W	H	D	w_i	h_i	d_i
1	100	100	100	$[1,\frac{1}{2}W]$	$[\frac{2}{3}H,H]$	$[\frac{2}{3}D,D]$
2	100	100	100	$[\frac{2}{3}W,W]$	$[1,\frac{1}{2}H]$	$[\frac{2}{3}D,D]$
3	100	100	100	$[\frac{2}{3}W,W]$	$[\frac{2}{3}H,H]$	$[1,\frac{1}{2}D]$
4	100	100	100	$[\frac{1}{2}W,W]$	$[\frac{1}{2}H,H]$	$[\frac{1}{2}D,D]$
5	100	100	100	$[1,\frac{1}{2}W]$	$[1,\frac{1}{2}H]$	$[1,\frac{1}{2}D]$
6	10	10	10	$[1,10]$	$[1,10]$	$[1,10]$
7	40	40	40	$[1,35]$	$[1,35]$	$[1,35]$
8	100	100	100	$[1,100]$	$[1,100]$	$[1,100]$

*The problem instances, together with their best known solution values, were downloaded from https://site.unibo.it/operations-research/en/research/2dpacklib

†The instance generator and the corresponding solver to determine the lower bound values are available at http://hjemmesider.diku.dk/~pisinger/codes.html

4.3 PERFORMANCE EVALUATION

This section describes the different performance metrics that are used to evaluate the CBS approach and compare it to the single-space searches and state-of-the-art techniques. The statistical tests that are used to determine the significance of the results are also discussed.

4.3.1 Performance Metrics

The bi-space search approach is compared to the single-space search approaches using three different performance metrics. The first metric is the number of bins in the solution found by the search technique for a given problem instance. This metric is selected because the problem statement for bin packing problems requires that the total number of bins used be a minimum. Chapter 5 (Eq. (4.1)) The second metric that is used is the cost function proposed by Falkenauer and Delchambre [71] which is a measure of the average bin efficiency, in other words how well the bins are packed or the total wasted space across all the bins in a given solution. The cost function used in this study is a modified version of the original so that it is a minimisation function. In addition, a value of one is added to the cost function for each unpacked item. This modification was made specifically for this study. Given that the search is allowed to operate on partial solutions, the modification was made so that complete solutions are favoured over partial ones. If two solutions being compared have the same number of items packed, then the solution with the better packing efficiency is favoured. If the two solutions have a different number of items packed, then the solution with more items packed (i.e. the one which is closest to being complete) is favoured. The value of the cost function is given by Eq. (4.1) below.

$$f_{\text{BPP}} = \left[1 - \frac{\sum_{i=1}^{N}(F_i/C)^2}{N}\right] + n \tag{4.1}$$

where N is the total number of bins used, F_i is the fullness (i.e. the occupied space (1D), area (2D) or volume (3D)) for the i^{th} bin, C is the total capacity of each bin and n is the number of items still to be packed. The third metric that is used is the computational time taken by the search technique to find a solution for a given problem instance. The CBS is compared to both SSS and HSS using each of these metrics, for each problem instance for each problem domain.

4.3.2 Statistical Tests

Hypothesis Test (One-Tailed z-Test)

The significance of the differences between the performance of the CBS approach and the single-space search approaches (for each of the three metrics used) is determined using a one-tailed z-test. The one-tailed z-test is used to determine whether the mean of one sample (e.g. sample A) is significantly less than the mean of a second sample (e.g. sample B). The test involves two hypotheses, namely the null hypothesis (H_0) which states that the means of the two samples are equivalent, and the alternative hypothesis (H_a) which states that the mean of sample A is less than the mean of sample B. The test statistic z (or *z-score*) is calculated using the equation:

$$z = \frac{\overline{B} - \overline{A}}{\frac{\sigma_A}{\sqrt{n_B}}} \tag{4.2}$$

where $\overline{A}$ and $\overline{B}$ are the means of samples A and B respectively, σ_A is the standard deviation of sample A and n_B is the size of sample B. If the z-score is greater than the critical value ($z > q_\alpha$) for a given confidence interval α then the null hypothesis can be rejected and the alternative hypothesis accepted. The critical values for α-levels of 1%, 5% and 10% (99%, 95% and 90% confidence respectively) are:

$$q_{0.01} = 2.33, \quad q_{0.05} = 1.96, \quad q_{0.10} = 1.65$$

4.4 TECHNICAL SPECIFICATIONS

The approach was developed on a computer with the following specifications: Intel(R) Core(TM) i5-8265U CPU @ 1.60 GHz, 16 GB RAM and 64-bit Windows 10 operating system. The approach was developed using Java 1.8 on the Netbeans 8.2 Integrated Development Environment (IDE). The simulations were run on the MITC cluster of the Department of Computer Science at the University of Pretoria, the specifications of which are: Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40 GHz running Ubuntu version 18.04 with 377 GB RAM.

CHAPTER 5 CONCURRENT BI-SPACE SEARCH

This chapter presents the details of the CBS approach proposed in this study. The chapter consists of four main sections. Section 5.1 describes the local search algorithm used in CBS and the single space searches to which CBS is compared. The perturbation operators for the solution space are then presented in Section 5.2 followed by the perturbation operators and low-level heuristics for the heuristic space in Section 5.3. Finally, the CBS approach is described in Section 5.4.

5.1 LOCAL SEARCH

This section describes the local search used both in the CBS approach and in the single space searches to which CBS is compared. First a general overview of the local search is presented and the processes that are common to all three searches are discussed. This is followed by a detailed description of each step in the local search and its application in each of the three searches.

Concurrently, exploring the two search spaces is anticipated to be computationally expensive as two search spaces are being explored. In light of the primary objective of this study, which is to demonstrate a proof of concept for CBS, a straightforward and cost-effective local search method is employed. This approach is not computationally demanding. Algorithm 1 outlines the local search procedure that is used [72].

Algorithm 1 Local Search (LS) [72]

```
1: Create initial solution s
2: for i ← 1,N do                          ▷ N = number of iterations
3:     s' ← perturb(s)                     ▷ perform a perturbation on s
4:     if acceptMove(s, s') then
5:         s ← s'                          ▷ update s for the next iteration
6: return s
```

The initial solution creation in Line 1 and perturbation in Line 3 are both dependent on the search space being explored. Both these aspects are discussed in detail in Section 5.2 for the solution space search (SSS), Section 5.3 for the heuristic space search (HSS) and Section 5.4 for the concurrent bi-space search (CBS). The move acceptance criterion in Line 4 is common to all three searches and is discussed below.

5.1.1 Move Acceptance Criteria

After the perturbation, a move acceptance criterion is applied to determine whether the perturbed solution s' should be accepted. If the decision is to accept s' then the current solution s is updated for the next iteration. These two steps are shown in Lines 4 and 5 respectively. Three different move acceptance criteria are investigated in this study: the *equal or improving* criterion, the *improving only* criterion, and the *adaptive iteration limited threshold accepting (AILTA)* [73] criterion. The AILTA criterion was selected because it has been shown to be effective for use in selection perturbative hyper-heuristics in other studies [74].

The equal or improving criterion accepts a perturbed solution if it is of equal quality or better than the unperturbed solution. Using a minimisation cost function this is represented mathematically as s' being accepted if $C(s') \leq C(s)$. The improving only criterion accepts perturbed solutions only if they are of better quality, which mathematically equates to accepting s' if $C(s') < C(s)$. AILTA is a dynamic move acceptance criterion, which allows the acceptance of worsening moves under certain conditions [73]. By allowing for the acceptance of worsening moves, AILTA escapes from local optima [73]. The pseudocode for AILTA is presented in Algorithm 2.

AILTA begins as an equal or improving move acceptance [73] as is shown in Lines 1 and 6. In Line 12 if the number of consecutive rejections of worsening moves has reached a specified number k, which is a parameter, then AILTA uses a threshold accepting criterion [73] as is shown in Line 13. Under threshold accepting a worse solution can be accepted if the difference in cost between the current and perturbed solutions is less than or equal to a specified threshold value t [73], as is shown in Line 13. If under threshold accepting a further j or more consecutive rejections, where j is a parameter, occurs then the threshold value is increased by an amount ε [73]. ε is a parameter to be tuned. This is shown in Lines 19 and 20. In Lines 21 and 22 an upper limit is imposed on the threshold value by the parameter t_{limit} [73].

Algorithm 2 Adaptive Iteration Limited Threshold Accepting (AILTA) [73]

Input: $k, j, \varepsilon, t_{start}, t_{limit}$

if $C(s') < C(s)$ **then**
    $s \leftarrow s'$
    $reject_{worse} = 0$
    $reject_{adapt} = 0$
    $t = t_{start}$
else if $C(s') = C(s)$ **then**
    $s \leftarrow s'$
    $reject_{adapt} = 0$
    $t = t_{start}$
else
    $reject_{worse} = reject_{worse} + 1$
    if $reject_{worse} \geq k$ **then**
        if $|C(s) - C(s')| \leq t$ **then**
            $s \leftarrow s'$
            $reject_{adapt} = 0$
            $t = t_{start}$
        else
            $reject_{adapt} = reject_{adapt} + 1$
            if $reject_{adapt} \geq j$ **then**
                $t = t + \varepsilon$
                if $t > t_{limit}$ **then**
                    $t = t_{limit}$

5.2 SOLUTION SPACE SEARCH

This section presents details of the local search that are specific to the solution space search. For local search in the solution space, a solution s is represented by a candidate solution to the BPP namely, a set of bins, with each bin containing items.

For all three problem domains the same initial solution is used, namely the empty solution in which no items are packed, and no bins are in use. Two perturbation operators are used in the solution space local search, one for 1BPP and another for both 2BPP and 3BPP. Both perturbation operators are implemented such that if the solution being perturbed has any unpacked items, it is decided at random whether to pack the next item using the best fit construction heuristic [28] or to apply the perturbation operator to the partial solution. The perturbation operator used for the 1BPP is discussed in Section 5.2.1.1 and the perturbation operator used for the 2BPP and 3BPP is discussed in Section 5.2.1.2.

5.2.1 Perturb Operators

This section describes the perturbation operators used in the solution space local search for each of these three problem domains. First, the perturbation operator for the 1BPP is presented, followed by the perturbation operators for the 2BPP and 3BPP. The same perturbation operator is used for both 2BPP and 3BPP. These operators were chosen because they were the best performing in the literature at the time of implementation for their respective problem domains [75, 76].

5.2.1.1 1BPP Solution Space Perturb Operator

This section outlines the perturbation operator that is used in the local search for 1BPP. The operator was adapted from the work of Levine and Ducatelle [75] and is outlined in Algorithm 3.

Algorithm 3 One-Dimensional Bin Packing Perturb Operator [75]

1: *free* $\leftarrow$ items from the least filled bin

2: **for** $i \leftarrow 1, n$ **do** $\triangleright$ n = number of remaining bins
3: Swap two items packed in the i^{th} bin with two items in *free* if the residual capacity of the bin is reduced after the swap

4: **for** $i \leftarrow 1, n$ **do** $\triangleright$ n = number of remaining bins
5: Swap two items packed in the i^{th} bin with one item in *free* if the residual capacity of the bin is reduced after the swap

6: **for** $i \leftarrow 1, n$ **do** $\triangleright$ n = number of remaining bins
7: Swap one item packed in the i^{th} bin with one item in *free* if the residual capacity of the bin is reduced after the swap

8: **while** *free* contains items **do**
9: remove the first item from *free* and pack it using the first fit construction heuristic [28]

The algorithm begins in Line 1 by selecting and removing the bin with the least amount of used space. The items from this bin are then added to a list called *free*. In Lines 2 and 3, the operator iterates over the bins that remain in the solution and attempts to find a feasible swap of *two packed* items with *two free* items such that there is less available space in the bin once the swap has been made. If a feasible swap is found the swap is made and the next bin is checked using the updated *free* list.

Once all bins have been checked, the operator proceeds to loop over the available bins again attempting to swap *two packed* items with *one free* item in the same manner as before. This is shown in Lines 4

and 5. As before, if a feasible swap is found the swap is made and the next bin is checked using the updated *free* list.

A third and final loop over the bins is made in Lines 6 and 7 attempting to swap *one packed* item with *one free* item. If a feasible swap is found the swap is made and the next bin is checked using the updated *free* list. After this final pass any items that are remaining in *free* are repacked using the first fit construction heuristic [28] as shown in Line 9.

Figure 5.1. Perturbation operator for 1BPP SSS.

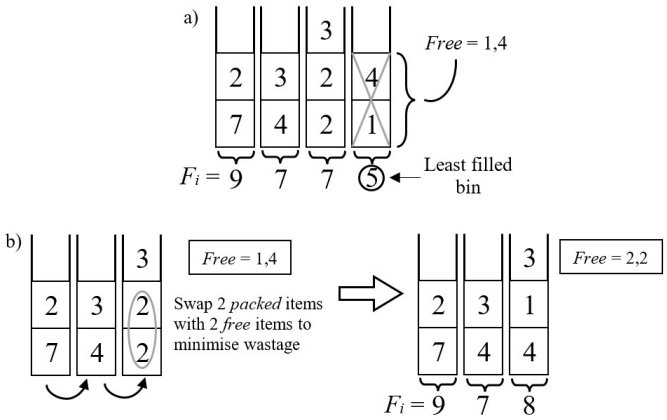


Figure 5.1 shows an example of the first step of the perturbation operator applied to a complete candidate solution s in the solution space for 1BPP. In figure a) the least-filled bin, containing the items of weights 1 and 4, is removed from the solution and its items are added to the list *free*. In figure b) the operator iterates over the remaining bins, finding a feasible swap in the third bin which contains two items with a weight of 2. In figure c) the swap is made leading to less wasted space in the third bin and the list *free* containing two items of weight 2.

5.2.1.2 2BPP & 3BPP Solution Space Perturb Operator

This section describes the perturbation operator that is used in the solution space local search for both the 2BPP and 3BPP. The operator was adapted from the work of Paschos [76]. The underlying principle of the operator is to iteratively remove items from the bin with the least amount of occupied space until the bin is empty and can be removed from the solution [76]. The pseudocode for the operator is presented in Algorithm 4.

Algorithm 4 Two & Three-Dimensional Bin Packing Perturb Operator [76]

```
1:  b ← least-filled bin
2:  K ← set of k randomly selected other bins
3:  I ← all items packed into bins in K

4:  while change = false ∧ items left in b do
5:      w ← next largest item in b
6:      I ← I ∪ {w_max}
7:      pack all items in I using at most k bins
8:      if packing found then
9:          remove w from least-filled bin b
10:         replace bins in K with new packing of I
11:         change ← true
```

The operator begins by selecting the least-filled bin b along with a set of k other random bins, shown in Lines 1 and 2. In Line 3, all items contained in the k selected bins are placed in list I. In Lines 5 and 6, the next largest item w is taken from bin b and added to list I. The operator then attempts to pack the items in I using a maximum of k bins as shown in Line 7. If a valid packing is found, then in Lines 9 and 10 the item w is removed from bin b and the k other bins are replaced with the new packing of I. If no valid packing is found, the process is repeated using the next largest item from bin b, as is shown in Line 5. If all items from bin b could not be packed into other existing bins and no feasible packing has been found, then the operator has no effect.

5.3 HEURISTIC SPACE SEARCH

This section presents details of the local search that are specific to the heuristic space search. The same representation, initialisation procedure and perturbation operator are used for all three problem domains. Each solution s in the heuristic space local search is represented as a sequence of LLHs, which determines the order in which each LLH is applied. In the case of 2BPP and 3BPP each LLH is a selection-placement pair, specifying the next item to be packed together with how it is to be placed into a given bin, in other words, the item's location within the bin.

The initial heuristic sequence is created as follows. The length of sequence l, i.e. the number of LLHs, is randomly selected from the range $[1, L_{\text{init}}]$, where L_{init} is a parameter to be tuned. The sequence is then iteratively constructed by randomly selecting an LLH from the set of all possible LLHs and appending it to the sequence. This process is repeated until the sequence has a specified length l. The set of heuristics from which each LLH is selected comprises of both constructive and perturbative

heuristics.

The heuristic space perturbation operator used for the three domains is implemented as follows: The operator begins by selecting a random subsequence of consecutive LLHs and removing them from the larger sequence. The length of this subsequence is randomly selected in the range $[0, H_{\text{repl}}]$, where H_{repl} is a parameter. A new replacement subsequence of LLHs is generated in the same manner as the initialisation procedure, where the length of this sequence is randomly selected from the range $[0, H_{\text{ins}}]$ with H_{ins} being a parameter. This newly generated subsequence is then inserted into the original sequence at the position of the removed subsequence. The values of both H_{repl} and H_{ins} can be zero, allowing the perturbation operator to perform replacement, insertion or deletion.

Figure 5.2. The perturbation operator for the 1BPP heuristic space local search.

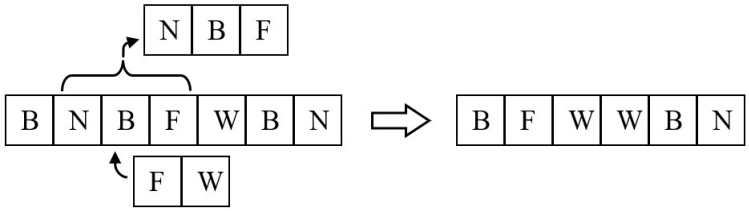


Figure 5.2 shows an example of the perturbation operator applied to the 1BPP LLH sequence $BNBFWBN$. First, the subsequence NBF (indices $1-3$) is randomly selected and removed from the string. A new replacement subsequence FW is then randomly generated and inserted into the string at the position of the removed subsequence NBF (index 1). The new, perturbed LLH sequence is $BFWWBN$.

The low-level heuristics selected for all three problem domains were chosen as they are the most commonly used in the literature for bin packing problems. The low-level construction heuristics used are outlined below [24, 28]. In all cases if all items in the solution have been packed then the heuristic has no effect.

- **First-Fit Decreasing/Increasing (FFD/I)**: The remaining items are sorted in descending /ascending order and the largest item is placed in the first feasible bin. If no such bin exists then the item is placed in a new bin.
- **Best-Fit Decreasing/Increasing (BFD/I)**: The remaining items are sorted in descending or ascending order and the largest item is placed into the bin with the minimum free space after the item is packed. If no feasible bin exists, then the item is placed in a new bin.

- **Next-Fit Decreasing/Increasing (NFD/I)**: The remaining items are sorted in descending or ascending order and the largest item is placed in the current bin if feasible, else in a new bin.
- **Worst-Fit Decreasing/Increasing (WFD/I)**: Similar to the *Best-Fit Decreasing/Increasing* heuristic, with the condition being to maximize the free space (rather than minimize).
- **Minimum Bin Slack (MBS)**: The set of remaining items is searched for a subset of items which completely fills a single bin. If no such subset exists, then the subset with the minimum free space (slack) is used.
- **Relaxed MBS (R-MBS)**: The *MBS* heuristic with an allowed slack (as opposed to a full bin).
- **Time-Bounded R-MBS (TBR-MBS)**: A time-limited variant of the *R-MBS* heuristic.

The 2BPP and 3BPP problem domains use the following additional constructive heuristics [16]:

- **Filler + FFD**: Attempts to place any of the remaining items into any of the currently open bins. Items are placed in descending order (largest area first). Each open bin is considered before placed the next item in the list. The process is stopped once at least one item has been packed. If no items can be packed then the FFD heuristic is applied.
- **Djang and Fitch (DJD)**: Beginning with the largest unpacked item and moving according to decreasing size items are placed into a new bin until the bin is at least one third full. An allowed waste w is then initialised and the heuristic searches for combinations of one, two or three remaining items that can be placed into the bin such that the wasted space (area) is at most w. If no such combination exists w is increased and the process is repeated.

For each of the low-level perturbative heuristics used the heuristic is applied only if the candidate solution has more than two bins. The rationale behind this choice is that any perturbations on solutions with two or fewer bins would undo the effects of the constructive heuristics applied up to that point. The low-level perturbative heuristics used are outlined as follows [36, 77].

- **Local Search Swap**: Swaps two randomly selected items from two different, randomly selected bins if there is space and if the swap leads to an improvement in the solution quality. If the swap does not lead to an improvement then the heuristic has no effect.
- **Mutation Swap**: Swaps two randomly selected items from two different, randomly selected bins if there is space. If one of the items does not fit, it is placed in a new bin.

- **Split a Bin**: Randomly selects a bin containing more items than the average number of items per bin and splits it into two new bins, each containing half of the items.
- **Repack Lowest Bin**: Delete the least-filled bin and repack its items using the *Best Fit* low-level construction heuristic.
- **Destroy x Highest Bins**: Delete the x most-filled bins, where x is a parameter, and repack their items using the *Best Fit Decreasing* heuristic. The heuristic only has an effect if the solution contains at least $x+2$ bins.
- **Destroy x Lowest Bins**: Delete the x least-filled bins, where x is a parameter, and repack their items using the *Best Fit Decreasing* heuristic. The heuristic only has an effect if the solution contains at least $x+2$ bins.

The following placement heuristics were used for 2BPP and 3BPP [58]:

- **Distance to Top (Front) Right Corner 1 (DTRC-1)**: The item is placed in a feasible position in the bin such that the distance from the (back) bottom left corner of the item to the top (front) right corner of the bin is maximized.
- **Distance to Top (Front) Right Corner 2 (DTRC-2)**: The item is placed in a feasible position in the bin such that the distance from the top (front) right corner of the item (after placement) to the top (front) right corner of the bin is maximized.

5.4 CONCURRENT BI-SPACE SEARCH (CBS)

This section presents the details of the local search that are specific to CBS. For all three problem domains the same representation and initialisation procedures are used. A local search solution for CBS consists of a candidate solution to the BPP, representing a point in the solution space search, and a sequence of LLHs, representing a point in the heuristic space search. As with the solution space search, the initial candidate solution for the bi-space is taken to be the empty solution. In the case of the initial sequence of LLHs, the same initialisation procedure described in Section 5.3 is used.

The local search for the CBS approach differs from that of the single space searches in that the perturbation step in Line 3 of Algorithm 1 involves a perturbation and move acceptance in both the heuristic and solution spaces. The pseudocode for the CBS perturbation operator is presented in Algorithm 5.

Algorithm 5 Concurrent Bi-Space Perturb Operator

input: heuristic space individual H, solution space individual S

1: $H' \leftarrow perturb(H)$ ▷ perturb H (and evaluate it on S)
2: **if** $acceptMove(H, H')$ **then** ▷ apply move acceptance criteria to H'
3: $H \leftarrow H'$

4: $S' \leftarrow$ evaluate H on S ▷ evaluate H on S
5: **if** $acceptMove(S, S')$ **then** ▷ apply move acceptance criteria to S'
6: $S \leftarrow S'$

7: $S' \leftarrow perturb(S)$ ▷ perform a perturbation on S
8: **if** $acceptMove(S, S')$ **then** ▷ apply move acceptance criteria to S'
9: $S \leftarrow S'$

10: **return** H, S

The operator begins by applying a perturbation and move acceptance to the point in the heuristic space H, as shown in Lines 1, 2 and 3. The perturbation operator applied is the same as that described above for the HSS. The operator then evaluates the current heuristic sequence H on the current solution S. This is shown in Line 4. A new solution S' is obtained. A move acceptance criterion is applied to S' and the current solution S is updated accordingly, as shown in Lines 5 and 6. Finally, in Lines 7, 8 and 9 the current solution S is perturbed and move acceptance is applied to the perturbed solution S' to determine whether to accept it or not.

CHAPTER 6 RESULTS AND DISCUSSION

This chapter presents the results of the computational experiments and a discussion thereof. The chapter begins with a presentation of the results and discussion for the experiments conducted to compare the CBS approach with the single-space searches HSS and SSS (Section 6.1). The results are presented, in order, for the 1BPP, 2BPP and 3BPP problem domains. An analysis of the CBS approach for each of the problem domains is then presented in Section 6.2. The sensitivity of CBS to the move acceptance criteria is analysed in Section 6.3. Section 6.4 presents a comparison of the CBS approach with the previous bi-space search approaches BSS and CSA. Finally, a comparison of the CBS approach with state-of-the-art techniques is presented in Section 6.5 for 1BPP, 2BPP and 3BPP.

6.1 COMPARISON OF CBS WITH HSS AND SSS

This section presents a comparison of the performance of the bi-space CBS approach to the single-space searches HSS and SSS. The comparisons are made using the number of bins in the solutions found by the searches, wastage formula values of the solutions found by the searches, and runtimes of the searches. The comparisons are made for each of the 1BPP, 2BPP and 3BPP problem domains. The z-test (described in Section 4.3.2) is used to determine the significance of the results.

6.1.1 One-Dimensional Bin Packing Problem

Table 6.1 shows the number of 1BPP problem instances that were solved to the optimum number of bins or one bin from the optimum for each of the three approaches (CBS, HSS and SSS), grouped according to the benchmark datasets (described in Section 4.2.1). The best results are reported in bold. From Table 6.1 it can be seen that the CBS approach consistently outperforms both the HSS and SSS approaches, except for two datasets. The first is the *Schwerin_1* dataset for which SSS is able to find

Table 6.1. Number of 1BPP problem instances solved to optimum. The best results are reported in bold.

Problem Set	Concurrent Bi-Space (CBS)			Heuristic Space (HSS)			Solution Space (SSS)		
	Opt.	Opt.-1	Sum	Opt.	Opt.-1	Sum	Opt.	Opt.-1	Sum
Falkenauer_T (80)	**1**	**79**	**80**	0	2	2	0	0	0
Falkenauer_U (80)	**50**	**30**	**80**	14	26	40	21	30	51
Scholl_1 (720)	**695**	**25**	**720**	625	52	677	616	84	700
Scholl_2 (480)	**463**	**16**	**479**	304	90	394	366	62	428
Scholl_3 (10)	**8**	**2**	**10**	0	0	0	1	3	4
Schwerin_1 (100)	97	3	100	16	84	100	**99**	**1**	**100**
Schwerin_2 (100)	**98**	**2**	**100**	65	35	100	**98**	**2**	**100**
Hard28 (28)	**5**	**23**	**28**	**5**	**23**	**28**	**5**	**23**	**28**
Wäscher (17)	**12**	**5**	**17**	5	12	17	9	8	17
Total (1615)	**1429**	**185**	**1614**	1034	324	1358	1215	213	1428

the optimum for two instances more than CBS. The second is the *Hard28* dataset for which all three approaches have equal performance.

Table 6.2. Average runtime (in seconds) per 1BPP problem instance for each dataset. The best results are reported in bold.

	CBS	HSS	SSS
Falkenauer_T (80)	8.543	4.263	**0.834**
Falkenauer_U (80)	17.753	10.967	**1.188**
Scholl_1 (720)	4.447	2.943	**0.403**
Scholl_2 (480)	3.263	1.241	**0.269**
Scholl_3 (10)	8.894	2.154	**0.565**
Schwerin_1 (100)	0.735	0.760	**0.155**
Schwerin_2 (100)	1.289	0.977	**0.129**
Hard28 (28)	5.745	2.375	**0.344**
Wäscher (17)	2.320	1.110	**0.229**

The average runtimes (in seconds) per problem instance, taken over each dataset, are reported in Table 6.2. From the table it can be seen that SSS is the quickest approach of the three, averaging under one second per problem instance for all datasets except *Falkenauer_U*, for which it is only slightly above one second. Comparing the times for CBS and HSS one can see that the two approaches tend to have similar runtimes, with the largest differences being for *Falkenauer_U*, where CBS averages approximately two seconds faster, and for *Scholl_3*, where CBS averages approximately two and a half seconds slower. As CBS is exploring two search spaces concurrently, it is anticipated that it have longer runtimes than either SSS or HSS. This is evident in Table 6.2. However, although CBS takes longer

to run than HSS or SSS, the average runtimes of CBS per problem instance are well within practical limits being under ten seconds for each dataset. This, combined with the performance increase over the single space searches evident in Table 6.1, indicates that CBS is a practical alternative to either HSS or SSS for solving 1BPP.

For each 1BPP problem instance the hypothesis test (z-test*) was performed to determine the statistical significance of the performance difference between CBS and HSS and SSS. The number of 1BPP problem instances where CBS outperformed the single space searches at 1 % statistical significance are reported in Table 6.3 for HSS and SSS using the number of bins and the wastage formula (see Section 5.2, Eq. (4.1)) as performance metrics.

Table 6.3. Number of 1BPP problem instances where CBS performed better than HSS and SSS at 1% significance.

	CBS < HSS	CBS < SSS
bins	994	697
wastage	1216	941

From the table it can be seen that the result that CBS found solutions with a lower number of bins than HSS was found to be statistically significant at the 1% level of significance for over 60% (994/1615) of problem instances. When compared to SSS, the result that CBS found solutions with a lower number of bins was statistically significant at the 1% level of significance for over 43% (697/1615) of problem instances. As the searches were guided by the value of the wastage formula, rather than the number of bins, a more appropriate comparison between the approaches would be to use the wastage formula, i.e. the packing efficiency. From Table 6.3 it can be seen that, when comparing on the packing efficiency, the results that the CBS approach found better solutions than HSS and SSS was found to be statistically significant for the majority of 1BPP problem instances. This, combined with the results reported in Table 6.1, indicates that CBS is a more effective approach than either HSS or SSS for solving 1BPP.

6.1.2 Two-Dimensional Bin Packing Problem

In order to be consistent with the literature the values reported in this section, both for the number of bins and the runtimes, are the averages taken across the ten instances in each subclass of 2BPP problem

*See Section 4.3.2 for a description of the statistical tests used.

instance. Table 6.4 compares the performance, in terms of the number of bins, for the CBS, HSS and

Table 6.4. Average number of bins for each subclass of 2BPP problem instance. The best results are reported in bold.

Class	Bin size	n	LB	CBS	HSS	SSS	Class	Bin size	n	LB	CBS	HSS	SSS
1	10x10	20	7.1	**7.1**	**7.1**	**7.1**	6	300x300	20	1.0	**1.0**	**1.0**	**1.0**
		40	13.4	**13.4**	**13.4**	13.6			40	1.5	**1.7**	**1.7**	1.9
		60	19.7	**20.0**	**20.0**	20.2			60	2.1	**2.1**	**2.1**	2.2
		80	27.4	**27.4**	27.5	27.7			80	3.0	**3.0**	**3.0**	**3.0**
		100	31.7	**31.7**	**31.7**	32.3			100	3.2	**3.4**	**3.4**	3.5
2	30x30	20	1.0	**1.0**	**1.0**	**1.0**	7	100x100	20	5.5	**5.5**	**5.5**	6.1
		40	1.9	**1.9**	**1.9**	2.0			40	10.9	11.2	**11.1**	11.5
		60	2.5	**2.5**	**2.5**	2.6			60	15.6	**15.8**	15.9	16.3
		80	3.1	**3.1**	**3.1**	3.3			80	22.4	**23.2**	**23.2**	**23.2**
		100	3.9	**3.9**	**3.9**	4.0			100	26.9	**27.1**	27.2	27.5
3	40x40	20	5.1	**5.1**	**5.1**	5.3	8	100x100	20	5.8	**5.8**	**5.8**	6.1
		40	9.2	**9.4**	**9.4**	9.8			40	11.2	11.4	**11.3**	11.5
		60	13.6	**13.9**	14.0	14.2			60	15.9	**16.1**	**16.1**	16.4
		80	18.7	**19.0**	19.1	19.5			80	22.3	**22.3**	22.5	22.7
		100	22.1	**22.3**	22.4	23.2			100	27.4	**27.7**	27.8	28.0
4	100x100	20	1.0	**1.0**	**1.0**	**1.0**	9	100x100	20	14.3	**14.3**	**14.3**	14.4
		40	1.9	**1.9**	**1.9**	**1.9**			40	27.8	**27.8**	**27.8**	28.0
		60	2.3	**2.5**	**2.5**	2.6			60	43.7	**43.7**	**43.7**	43.9
		80	3.0	**3.1**	3.2	3.3			80	57.7	**57.7**	**57.7**	57.8
		100	3.7	**3.8**	**3.8**	3.9			100	69.5	**69.5**	**69.5**	69.6
5	100x100	20	6.5	**6.5**	**6.5**	6.6	10	100x100	20	4.2	**4.2**	**4.2**	4.4
		40	11.9	**11.9**	**11.9**	12.3			40	7.4	**7.4**	**7.4**	**7.4**
		60	17.9	**18.0**	**18.0**	18.5			60	9.8	10.3	**10.1**	10.5
		80	24.1	**24.7**	**24.7**	25.2			80	12.3	**13.0**	**13.0**	13.2
		100	27.9	**28.3**	**28.3**	29.0			100	15.3	**15.9**	**15.9**	16.3

SSS approaches for 2BPP. From the table it can be seen that CBS outperforms SSS for 42 out of the 50 problem subclasses and has equal performance for the remaining 8. CBS outperforms HSS for 9 out of the 50 subclasses and is equal for a further 38, being outperformed for only 3 subclasses.

The average runtime per problem instance for each of the three approaches is shown in Table 6.5. From the table it can be seen that SSS is consistently the fastest approach, never averaging above one second. Comparing the runtimes of CBS with those of HSS, it can be seen from the table that CBS has lower runtimes than HSS for all but five subclasses. However, for those five subclasses the difference in times between the two approaches is less than 0.1 seconds. As with 1BPP, if one considers the average runtimes of CBS in isolation, the runtimes are well within feasible limits, with the highest average runtime being only five and a half seconds and the average across all problem instances being a mere 1.8 seconds. Given that CBS outperforms SSS and has approximately equivalent performance

Table 6.5. Average runtime (in seconds) per 2BPP problem instance for each subclass of problem instance. The best results are reported in bold.

Class	Bin size	n	CBS	HSS	SSS	Class	Bin size	n	CBS	HSS	SSS
1	10x10	20	0.356	1.295	**0.044**	6	300x300	20	0.015	**0.007**	**0.007**
		40	0.692	2.388	**0.071**			40	2.364	7.980	**0.034**
		60	1.452	4.072	**0.063**			60	0.129	1.691	**0.106**
		80	2.139	6.296	**0.073**			80	0.108	0.041	**0.028**
		100	1.986	7.421	**0.084**			100	3.337	94.296	**0.393**
2	30x30	20	0.010	0.004	**0.001**	7	100x100	20	0.788	1.846	**0.081**
		40	0.073	0.417	**0.016**			40	2.248	7.517	**0.267**
		60	0.089	1.782	**0.044**			60	3.079	20.425	**0.369**
		80	0.239	7.718	**0.133**			80	3.180	23.465	**0.441**
		100	0.165	7.482	**0.099**			100	4.337	77.100	**0.684**
3	40x40	20	0.568	1.944	**0.055**	8	100x100	20	0.774	1.727	**0.074**
		40	1.266	4.817	**0.256**			40	2.182	7.726	**0.434**
		60	2.380	19.642	**0.837**			60	3.074	40.608	**0.457**
		80	3.106	33.057	**0.635**			80	3.567	19.861	**0.446**
		100	3.364	53.872	**0.521**			100	5.574	55.051	**0.752**
4	100x100	20	0.014	0.007	**0.003**	9	100x100	20	0.861	2.179	**0.091**
		40	0.049	0.034	**0.017**			40	0.945	2.523	**0.182**
		60	2.088	13.107	**0.158**			60	1.532	3.031	**0.260**
		80	3.057	38.682	**0.258**			80	1.949	3.896	**0.310**
		100	1.575	59.190	**0.241**			100	2.584	5.418	**0.261**
5	100x100	20	0.753	2.409	**0.076**	10	100x100	20	0.814	1.954	**0.064**
		40	1.792	6.076	**0.303**			40	0.907	7.635	**0.116**
		60	2.389	14.487	**0.472**			60	2.148	26.345	**0.428**
		80	2.776	12.103	**0.361**			80	2.903	58.069	**0.858**
		100	4.518	94.444	**0.482**			100	3.087	124.591	**0.986**

to HSS (as shown in Table 6.4), and that CBS has practical runtimes, it can be concluded that CBS is an effective alternative to either HSS or SSS.

As with 1BPP, z-tests were performed for each problem instance, using the number of bins and the wastage formula/packing efficiency as performance metrics. Table 6.6 shows the number of 2BPP problem instances where CBS had a lower number of bins and wastage than HSS and SSS. These results were found to be statistically significant at the 1% level of significance.

Table 6.6. Number of 2BPP problem instances where CBS performed better than HSS and SSS at 1% significance.

	CBS < HSS	CBS < SSS
bins	28	132
efficiency	122	306

Table 6.6 shows that for 2BPP CBS had a lower number of bins than the HSS for only 28 instances out of 500, and a better packing efficiency for a quarter of the total problem instances. These results were statistically significant at the 1% level of significance. These small numbers are not unsurprising given that Table 6.4 shows that the CBS and HSS approaches have similar performance. However, as has been noted in the discussion of Table 6.5, CBS has lower runtimes than HSS, making it a more practical approach to 2BPP than HSS. Comparing the CBS approach to SSS, Table 6.6 shows that CBS had a lower number of bins for approximately 25% of the problem instances and a better packing efficiency for over 60% of the problem instances. These results were statistically significant at the 1% level of significance. As before, with the packing efficiency being used to guide the searches, it is a better indicator of the relative performance between the two approaches. The results of the z-tests, coupled with the observations made in the discussions of Tables 6.4 and 6.5, indicate that CBS is a more practical approach to 2BPP than either HSS or SSS.

6.1.3 Three-Dimensional Bin Packing Problem

To remain consistent with the literature, the values reported in this section are the averages across the ten instances in each subclass of 3BPP problem instance. Table 6.7 compares the performance of the CBS, HSS and SSS approaches for each subclass of 3BPP problem instance. As is done in the literature, the values reported are the ratio of the number of bins to the lower bound value $L2$ for each problem instance. The lower bound values were taken from the literature [55]. From the table it can be seen that CBS outperforms both single space searches for 16 out of the 32 problem subclasses and is tied with HSS for the best for a further nine subclasses. HSS outperforms CBS for the remaining seven problem subclasses. Considering the average across all problem classes shows that CBS is the best of the three approaches.

The average runtimes, across the ten instances per 3BPP problem class, for each of the three approaches are shown in Table 6.8. From the table it can be seen that, as with 1BPP and 2BPP, SSS has the lowest runtimes for all problem classes, with CBS having runtimes significantly lower than HSS. Although CBS has longer runtimes than SSS, the runtimes are still within feasible limits, with the highest average runtime being approximately seven minutes (for class five, $n = 200$). For the same class of problem instance HSS is more than 60 times slower than CBS and SSS is less than nine times faster than CBS. For the seven problem classes for which HSS outperformed CBS (reported in Table 6.7) HSS ranges from 25 times slower than CBS (class two, $n = 150$) to over 187 times slower (class six, $n = 200$).

Table 6.7. Average number of bins relative to the lower bound for each class of 3BPP problem instance. The best results are reported in bold.

Class	Bin Size	n	L2	CBS / L2	HSS / L2	SSS / L2
1	100x100x100	50	12.5	**1.036**	**1.036**	1.054
		100	23.2	**1.056**	1.060	1.086
		150	37.2	**1.050**	1.061	1.084
		200	47.6	**1.042**	1.055	1.078
2	100x100x100	50	12.4	**1.040**	1.040	1.080
		100	24.3	**1.062**	1.070	1.095
		150	36.0	1.062	**1.059**	1.090
		200	47.2	**1.034**	1.041	1.058
3	100x100x100	50	12.5	**1.033**	**1.033**	1.063
		100	23.1	**1.085**	1.089	1.113
		150	34.8	**1.047**	1.067	1.087
		200	48.8	**1.048**	1.060	1.082
4	100x100x100	50	31.2	**1.000**	**1.000**	**1.000**
		100	57.0	**1.026**	**1.026**	1.033
		150	87.4	**1.023**	1.028	1.036
		200	113.5	**1.024**	1.025	1.032
5	100x100x100	50	7.9	**1.062**	**1.062**	1.105
		100	12.0	**1.170**	**1.170**	1.218
		150	16.0	1.149	**1.138**	1.190
		200	22.0	**1.143**	1.147	1.175
6	10x10x10	50	10.5	**1.000**	1.017	1.067
		100	18.0	**1.078**	1.092	1.126
		150	26.7	1.094	**1.092**	1.109
		200	36.3	1.075	**1.072**	1.094
7	40x40x40	50	5.7	**1.144**	**1.144**	1.215
		100	12.1	**1.162**	**1.162**	1.190
		150	15.7	**1.132**	1.144	1.189
		200	20.4	1.158	**1.145**	1.186
8	100x100x100	50	9.4	**1.013**	**1.013**	1.094
		100	17.2	**1.079**	1.094	1.132
		150	20.4	1.131	**1.125**	1.176
		200	27.1	1.164	**1.149**	1.187

Taking the average over all 320 problem instances, CBS has an average runtime of under a minute whilst HSS has an average runtime of over 53 minutes. Given that the runtimes of CBS are within practical limits and are orders of magnitude faster than HSS, together with the superior performance of CBS (as shown in Table 6.7), it can be concluded that CBS is a practical and effective alternative to either HSS or SSS for 3BPP.

Z-tests were performed for each problem instance to determine the statistical significance of the

Table 6.8. Average runtime in seconds for each class of 3BPP problem instance. The best results are reported in bold.

Class	Bin Size	n	CBS	HSS	SSS	Class	Bin Size	n	CBS	HSS	SSS
1	100x100x100	50	5.55	24.03	**2.06**	5	100x100x100	50	6.58	131.19	**2.08**
		100	16.62	348.72	**2.56**			100	49.52	1607.38	**20.21**
		150	44.54	696.14	**2.76**			150	185.15	9657.05	**36.91**
		200	50.55	1719.08	**2.89**			200	428.39	26247.29	**49.71**
2	100x100x100	50	5.04	53.67	**1.69**	6	10x10x10	50	2.37	25.13	**0.75**
		100	20.13	290.22	**2.51**			100	4.98	303.05	**0.58**
		150	45.83	1165.86	**3.41**			150	4.18	678.98	**0.65**
		200	63.14	2533.60	**3.79**			200	7.16	1342.90	**0.14**
3	100x100x100	50	3.94	31.08	**1.47**	7	40x40x40	50	5.96	168.66	**1.65**
		100	12.78	407.51	**2.24**			100	41.95	1159.66	**9.41**
		150	36.85	727.05	**2.65**			150	168.30	5105.30	**25.36**
		200	43.58	3461.12	**4.68**			200	306.56	26394.42	**30.01**
4	100x100x100	50	1.33	5.68	**0.36**	8	100x100x100	50	5.11	73.24	**1.44**
		100	3.12	19.54	**0.37**			100	20.45	500.36	**3.56**
		150	7.09	29.62	**0.58**			150	101.15	5146.70	**9.21**
		200	11.66	85.66	**0.76**			200	137.85	11890.79	**13.50**

difference in performance between the CBS, HSS, and SSS approaches. The number of bins and wastage formula (i.e. packing efficiency) were used as performance measures. The number of problem instances for which CBS performed better than HSS and SSS measured in terms of the number of bins and packing efficiency at the 1% level of significance is shown in Table 6.9.

Table 6.9. Number of 3BPP problem instances where CBS performed better than HSS and SSS at 1% significance.

	CBS < HSS	CBS < SSS
bins	13	154
efficiency	141	272

From the table it can be seen that CBS outperformed HSS for 13 problem instances in terms of the number of bins and for 141 problem instances in terms of packing efficiency. From Table 6.7 one can see that the low number of instances when comparing CBS and HSS on the number of bins is not unexpected as the two approaches have similar results when the number of bins is used. However, when comparing the two approaches by packing efficiency, CBS outperforms HSS for over 44% of the problem instances. When coupling these observations with the difference in runtimes between the two approaches, one can conclude that CBS is a practical and effective alternative to HSS for 3BPP.

For the case of SSS, CBS performed significantly better than SSS for almost 50% (154 out of 320) of the problem instances when comparing solutions according to the number of bins, and for over 85%

(272 out of 320) of the problem instances when comparing solutions according to packing efficiency. Although SSS has lower runtimes than CBS, as shown in Table 6.8, the fact that CBS outperforms SSS for almost all 3BPP problem instances, shown in Table 6.7, and that the difference is statistically significant for the majority of problem instances, as shown in Table 6.9, indicates that CBS is a practical and effective alternative to SSS for 3BPP.

The CBS approach is compared with the single-space searches HSS and SSS in terms of the number of bins, packing efficiency and runtimes for each of the 1BPP, 2BPP, and 3BPP problem domains. Whilst SSS consistently has the lowest runtimes across the three domains, CBS and HSS frequently find better solutions, both in terms of the number of bins and packing efficiency. For all three domains CBS finds, on average, equal or better solutions than HSS. In cases where CBS finds equal quality solutions to HSS, it does so in less time, making it the more effective choice.

6.2 ANALYSIS OF THE APPROACH

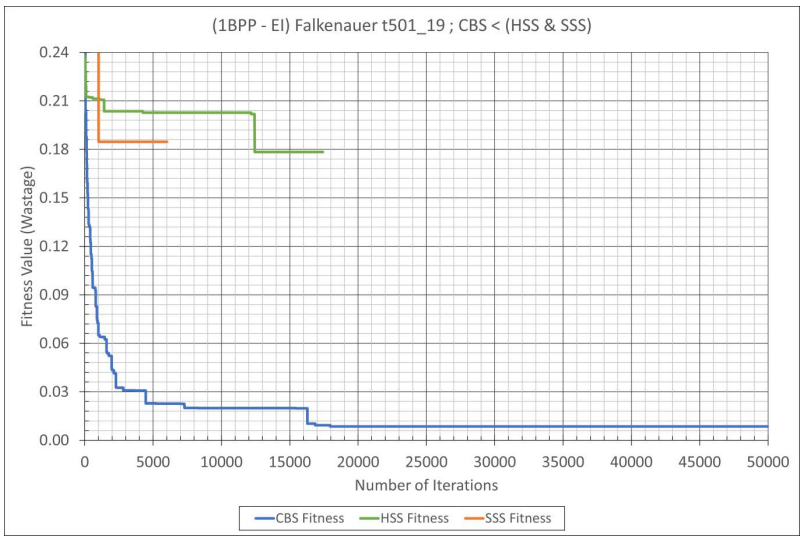
This section presents an analysis of the CBS approach for each of the 1BPP, 2BPP and 3BPP problem domains. An analysis of how the CBS approach moves through the search space, relative to the single-space searches HSS and SSS, is conducted by means of plots of the packing efficiency curves of the searches. The aim is to examine convergence of the CBS compared to SSS and HSS when it performs better and when it performs worse than these searches to better explain its performance.

Each of the figures presented in this section show the change in wastage values over the duration of the search for the SSS, HSS and CBS approaches. The results shown are for the best out of 30 runs, using the *equal or improving (EI)* move acceptance criterion (described in Section 5.1.1). The instances were selected such that, for each of the three domains, there is an instance where CBS performs better than both HSS and SSS, an instance where CBS performs worse than HSS and an instance where CBS performs worse than SSS. These cases were verified by the z-test for each problem instance.

6.2.1 CBS Performing Better than HSS and SSS

Figure 6.1 shows the change in wastage values over the duration of the search (number of iterations) for the 1BPP Falkenauer_T *t501_19* problem instance, where CBS performed better than both HSS and SSS. From the figure it can be seen that, once all items have been packed, i.e. the wastage value is less than one, SSS gets stuck in a local optimum and is unable to escape, thus converging prematurely. HSS

Figure 6.1. Change in packing efficiency for the 1BPP Falkenauer_T *t501_19* problem instance.



begins by making a large improvement, which is then followed by several smaller improvements. A moderately large improvement is then made (at ∼1500 iterations) taking the search to a local optimum. Although this new local optimum is worse than that found by SSS, HSS is able to escape its optimum after some time (∼10000 iterations) and move the search (by means of a large improvement) to a new local optimum, one which is better than that found by SSS. However, HSS is unable to escape its new optimum and converges prematurely. In contrast, CBS makes several relatively large, consecutive improvements for approximately the first 5000 iterations, at which point the search gets stuck in a local optimum. After a further 10000 iterations CBS is able to escape this local optimum by making a moderate improvement, thereby moving the search to a new local optimum. CBS is unable to escape this new local optimum and converges prematurely. It should be noted that the local optimum at which CBS converges to is of better quality than that found by either SSS or HSS. Additionally, SSS and HSS converged prematurely quicker than CBS.

Figure 6.2 shows the change in wastage values over the number of iterations for the *cl_04_080_07* 2BPP problem instance, where CBS performed better than both HSS and SSS. From the figure it can be seen that once all items have been placed SSS gets stuck in a local optimum from which it is unable to escape, converging prematurely. HSS is able to make some initial small improvements before getting stuck in a local optimum. After some time HSS is able to escape this local optimum to a new slightly better optimum from which it is unable to escape. In contrast, CBS makes several small improvements before getting stuck in a local optimum (which is worse than the initial local optimum found by HSS). However, CBS is able to quickly escape to a global optimum.

Figure 6.2. Change in packing efficiency for the 2BPP *cl_04_080_07* problem instance, under *equal or improving* move acceptance.

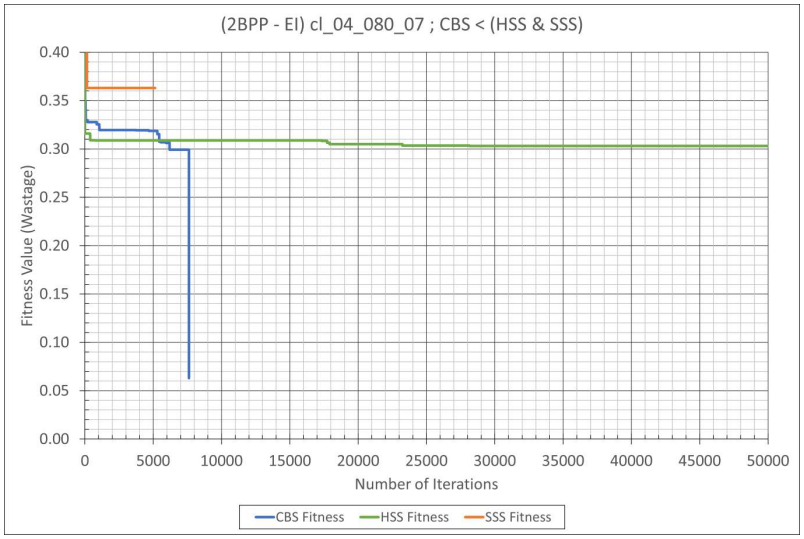
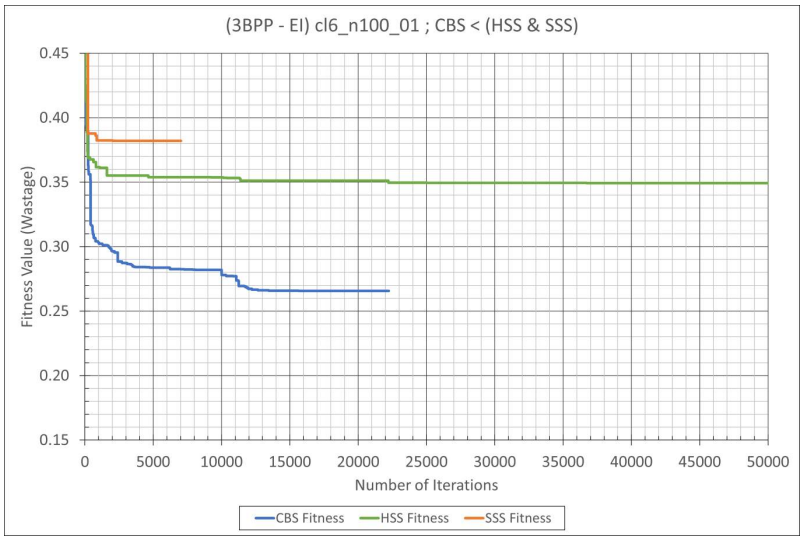


Figure 6.3. Change in packing efficiency for the 3BPP *cl6_n100_01* problem instance, under *equal or improving* move acceptance.



The change in wastage values over the number of iterations are shown in Fig. 6.3 for the 3BPP *cl6_n100_01* problem instance, where CBS performed better than both HSS and SSS. From the figure it can be seen that SSS makes a small improvement after which it gets stuck in a local optimum and converges prematurely (within 10000 iterations). HSS starts with a better fitness value than SSS and makes several small improvements before making a comparatively large improvement early on in the search (within 2500 iterations). HSS continues to make several small improvements over the duration of the search before getting stuck in a local optimum. CBS initially makes a large improvement in the

fitness, surpassing both SSS and HSS, followed by consecutive and increasingly smaller improvements until it gets stuck in a local optimum (at ∼2500 iterations). CBS is able to escape this local optimum (at ∼10000 iterations) making further small improvements before getting stuck in a new local optimum at which it converges prematurely.

6.2.2 CBS Performing Worse Than HSS

Figure 6.4. Change in packing efficiency for the 1BPP Scholl_1 *N3C2W2_N* problem instance.

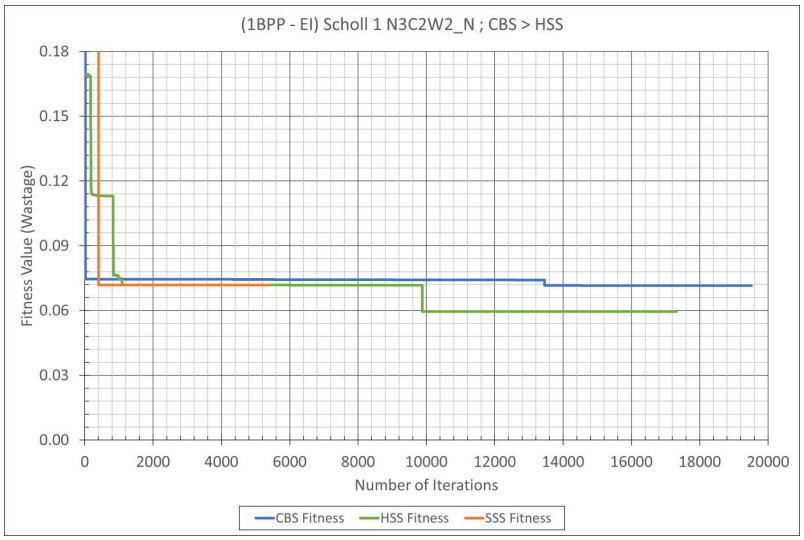


Figure 6.4 reports the change in wastage values over time for the 1BPP Scholl_1 *N3C2W2_N* problem instance. For this instance CBS performed worse than HSS. The figure shows that, as before, once all items have been packed, the SSS approach gets stuck in a local optimum from which it is unable to escape, before converging prematurely. The CBS approach begins with a single very large improvement, taking the search to a local optimum, which is worse than that at which SSS converged to. After approximately 13500 iterations, CBS makes a small improvement enabling it to move the search to a new local optimum, one which is of equal quality to that found by SSS. CBS is unable to escape this new local optimum and converges prematurely. In contrast, HSS makes two large improvements early on in the search (within 1000 iterations), taking it to a local optimum of equal quality to that found by SSS and CBS. At approximately 10000 iterations HSS makes a moderate improvement, enabling it to escape to a new local optimum, one which is of better quality than that found by SSS and CBS. HSS is unable to escape this new local optimum and converges prematurely. HSS and SSS converge prematurely quicker than CBS, however HSS converges to a better value than either SSS or CBS.

Figure 6.5. Change in packing efficiency for the 2BPP *cl_10_060_03* problem instance, under *equal or improving* move acceptance.

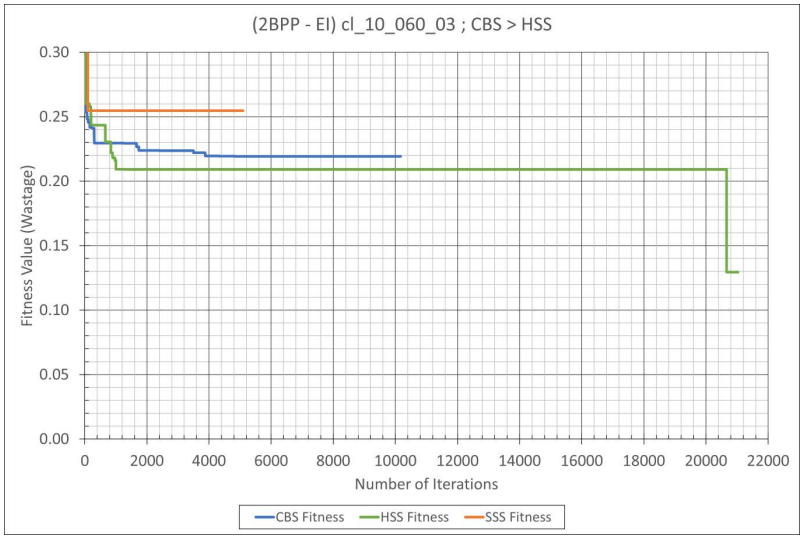
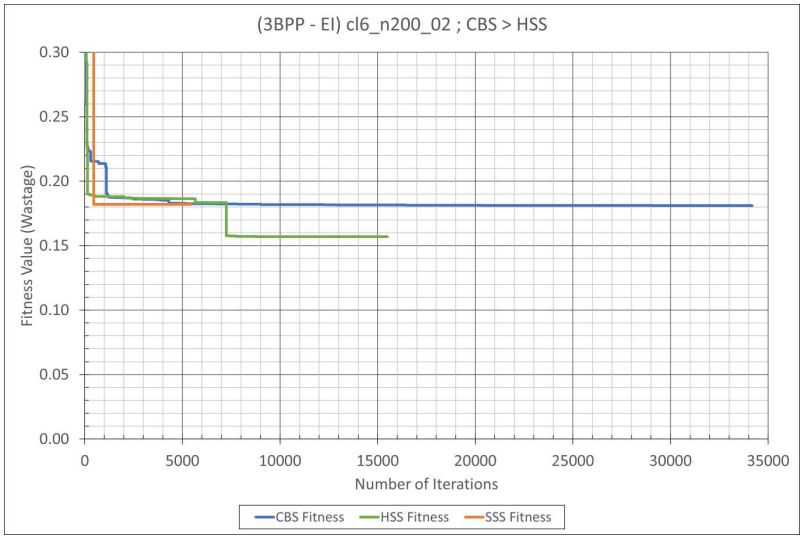


Figure 6.5 shows the change in fitness values over the duration of the search for the 2BPP *cl_10_060_03* problem instance. CBS performed worse than HSS for this problem instance. The figure shows that, once again, SSS gets stuck in a local optimum, once all items have been packed, from which it is unable to escape. CBS makes rapid initial improvements before quickly getting stuck in a local optimum. It is able to quickly escape this optimum before getting stuck in another local optimum. It is again able to quickly escape the local optimum before getting stuck in a final local optimum from which it is unable to escape. HSS makes several rapid improvements initially before getting stuck in a local optimum (of better quality than that of HSS). It remains stuck in this optimum for a long period of time before finally escaping to a new optimum from which it is unable to escape.

The change in fitness values over the duration of the search for the 3BPP problem instance *cl6_n200_02* are shown in Fig. 6.6. CBS performed worse than HSS for this problem instance. The figure shows that in the case of SSS, once all items have been packed (i.e. fitness value < 1.0) the search becomes stuck in a local optimum and is unable to escape. The CBS approach initially makes several small improvements before making a large improvement (at $\sim$1000 iterations). The search then continues to make several small improvements before getting stuck in a local optimum at approximately the same fitness value as SSS. In contrast, the HSS approach makes a large initial improvement, taking it to approximately the same value at which both SSS and CBS converged. However, at approximately 7000 iterations HSS makes a large improvement, escaping to a new, better local optimum at which it converges prematurely.

Figure 6.6. Change in packing efficiency for the 3BPP *cl6_n200_02* problem instance, under *equal or improving* move acceptance.



6.2.3 CBS Performing Worse Than SSS

Figure 6.7. Change in packing efficiency for the 1BPP Scholl_2 *N4W2B3R3* problem instance.

Figure 6.7 shows the change in wastage values over the number of iterations for the 1BPP Scholl_2 *N4W2B3R3* problem instance. For this problem instance, CBS performed worse than SSS. From the figure it can be seen that both CBS and HSS initially make relatively large improvements in the wastage, with the improvements becoming smaller over time, until the searches reach local optima. The HSS approaches reaches this optimum at approximately 4000 iterations, but soon escapes to a new local optimum of better quality (at $\sim$7000 iterations). The HSS approach is unable to make further improvements and converges prematurely. CBS begins similarly to HSS, making several improvements

before reaching a local optimum at approximately 5000 iterations. This optimum is of worse quality than that initially found by HSS. CBS remains stuck in this local optimum for a long period of time (∼17000 iterations) before finally escaping to a new local optimum (at ∼22000 iterations), at which it again becomes stuck for a long period, only being able to escape to another optimum after ∼24000 iterations. CBS remains at this optimum, which is of equal quality to that found by HSS, until the search terminates. SSS on the other hand is able to quickly find the global optimum solution (within ∼1000 iterations).

Figure 6.8. Change in packing efficiency for the 2BPP *cl_01_060_09* problem instance, under *equal or improving* move acceptance.

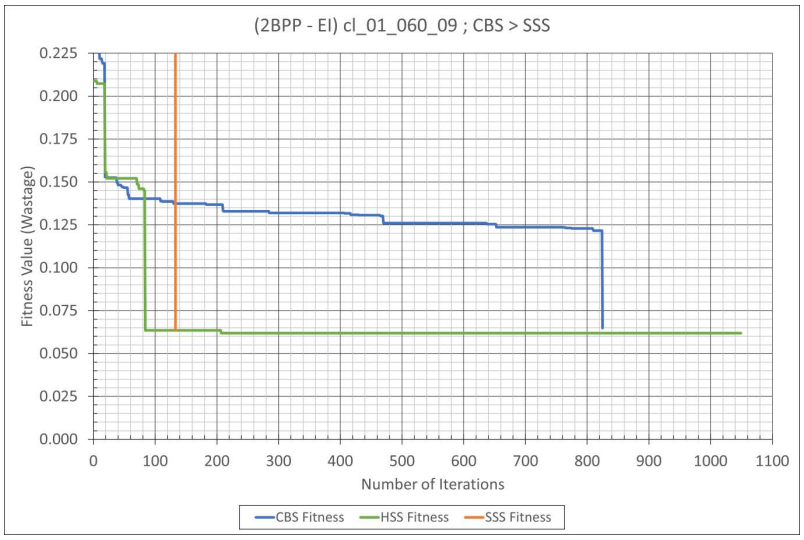
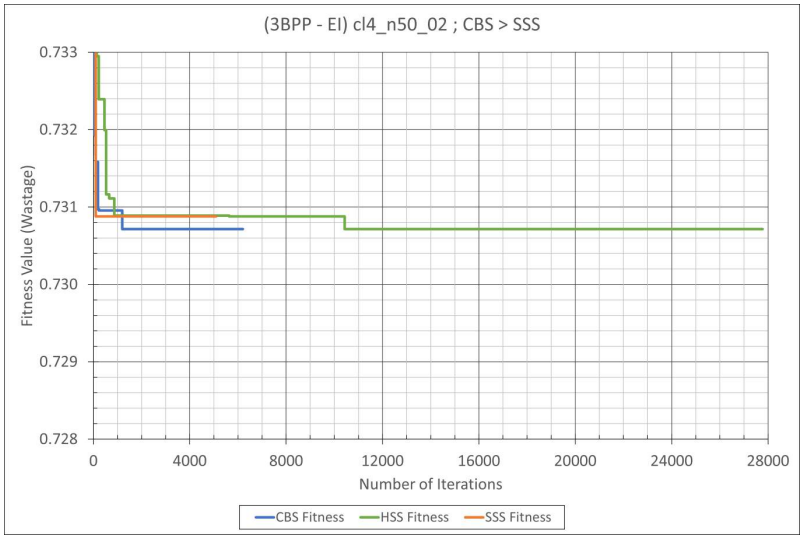


Figure 6.8 shows the change in wastage values over the duration of the search for the 2BPP *cl_01_060_09* problem instance. CBS performed worse than SSS for this problem instance. From the figure it can be seen that both CBS and HSS make rapid initial improvements. HSS briefly gets stuck in a local optimum early on (within 50 iterations) but is able to quickly escape to a better local optimum from which it cannot escape. CBS, on the other hand, continues to make smaller gradual improvements until it finds the global optimum. SSS in contrast rapidly finds the global optimum (within 150 iterations) once all items have been packed.

Figure 6.9 shows the changes in wastage values for the 3BPP *cl4_n50_02* problem instance. CBS performed worse than SSS for this problem instance. It is worth noting that whilst the z-test showed CBS to have a significantly higher fitness value than SSS for this problem instance, there was no 3BPP problem instance for which the best run using SSS performed better than the best run using CBS. For SSS the figure shows a similar pattern to that of the *cl6_n200_02* problem instance in

Figure 6.9. Change in packing efficiency for the 3BPP *cl4_n50_02* problem instance, under *equal or improving* move acceptance.



that the search becomes stuck in a local optimum and converges prematurely once all the items have been assigned. HSS makes several large improvements within the first 1000 iterations, reaching the same fitness value at which SSS converged. At approximately 10500 iterations HSS escapes by making a small improvement to get to a new local optimum at which it remains stuck. CBS makes one large improvement early on in the search, reaching a local optimum which it soon escapes (after approximately 1000 iterations). In escaping, CBS reaches a new local optimum at which it remains, converging prematurely. It should be noted that both HSS and CBS converge to the same fitness value, with CBS reaching this value significantly faster than HSS.

6.2.4 Summary of Analyses

For the chosen problem instances where CBS performs better than both HSS and SSS, Figs. 6.1, 6.2 and 6.3 show that the CBS approach makes large initial improvements, avoiding getting stuck in poor quality local optima early on in the search. In contrast, for the same problem instances the HSS and SSS approaches get stuck in local optima early on in the search.

For the problem instances where CBS performs worse than HSS or SSS the figures show that the CBS approach gets stuck in local optima early on in the search and has difficulty escaping, frequently converging prematurely. For the same problem instances, HSS and SSS also get stuck early on in the search but are able to escape sooner than CBS.

6.3 CBS SENSITIVITY TO MOVE ACCEPTANCE CRITERIA

This section investigates the sensitivity of CBS to different move acceptance criteria. It is not necessary to test the sensitivity of CBS to different perturbation operators as the operators used are the best known operators for bin packing cited in the literature. These operators are discussed in Section 5.2.1. The performance of three move acceptance criteria commonly used in the literature is compared for CBS for each of the problem instances used in Section 6.2. These criteria are the *equal or improving* (EI), *only improving* (OI) and *adaptive iteration limited threshold accepting* (AILTA) [73]. The problem instances were chosen both to have a diverse set of characteristics as well as to have an instance where CBS performed better than both HSS and SSS, an instance where CBS performed worse than HSS and an instance where CBS performed worse than SSS for each of the three problem domains. The values shown for all figures in this section are for the best out of 30 runs.

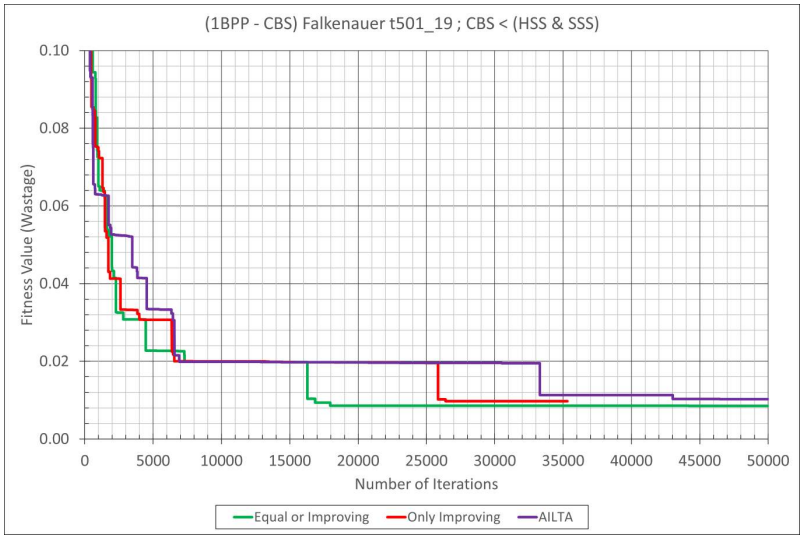
6.3.1 Overview of Move Acceptance Criteria

As described in Section 5.1.1, the EI criterion accepts moves that lead to solutions of equal or better quality, whilst the OI criterion only accepts moves that result in solutions of strictly better quality. The AILTA criterion begins as an EI acceptance, but switches to a threshold accepting after a specified number of consecutive rejections [73]. A more in-depth explanation of the AILTA criterion, together with pseudocode for the procedure, is provided in Section 5.1.1.

6.3.2 Move Acceptance Criteria - 1BPP

Figure 6.10 shows the change in wastage values for the Falkenauer_T *t501_19* problem instance. The figure shows the run using *EI* to perform better than either the *OI* or *AILTA* runs. From the figure it can be seen that for all three move acceptance criteria the CBS approach makes several large improvements early on in the search, before reaching a local optimum (at $\sim$7000 iterations). It is interesting to note that all three acceptance criteria reach the same local optimum at approximately the same time. The *AILTA* run takes the longest to escape this local optimum, doing so at approximately 33000 iterations through a large improvement to the wastage value. In escaping, the *AILTA*-based search reaches a new local optimum from which it escapes (after $\sim$10000 iterations) by means of a small improvement. The CBS remains at this wastage value for the remainder of the search. The *AILTA* run was stopped after 50000 iterations in order to limit the computational cost, however it can be seen that the search was still making improvements and likely will have converged to a better solution had it been allowed to

Figure 6.10. Change in packing efficiency for the 1BPP Falkenauer_T *t501_19* problem instance, for different move acceptance criteria.



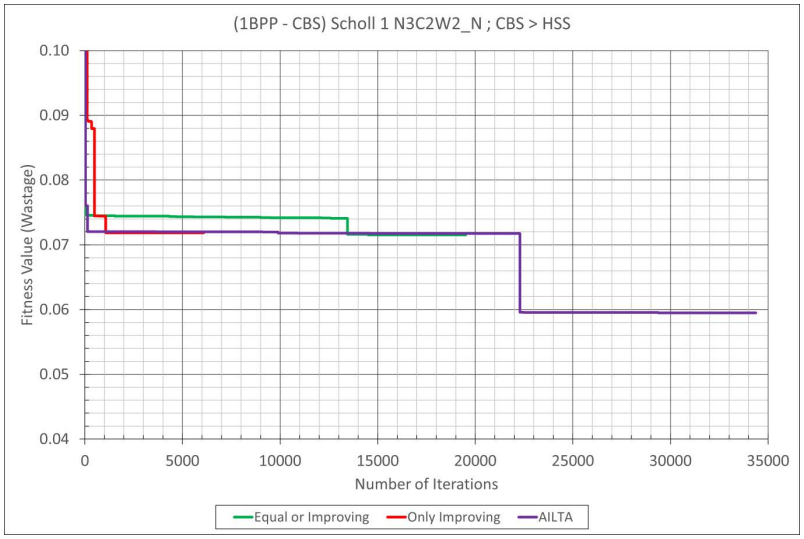
continue.

The run employing the *OI* move acceptance criterion escapes the first local optimum (which all three runs initially converge to) at approximately 26000 iterations (∼7000 iterations sooner than the *AILTA* run) by means of a large improvement to the wastage value. This improvement takes the search to a new local optimum at which it converges prematurely. It is to be noted that the final wastage values of the *AILTA* and *OI* runs are of equal quality.

The *EI* run escapes the first optimum the quickest, doing so at approximately 16000 iterations. This is achieved through a large improvement, which is followed by two small improvements, leaving the search at a new local optimum. The CBS approach remains at this optimum for the remainder of the search. It should be noted that the final wastage value of the *EI* run is of better quality than that of the *AILTA* and *OI* runs.

The changes in wastage values for the 1BPP Scholl_1 *N3C2W2_N* problem instance are shown in Fig. 6.11, from which it can be seen that the run employing *AILTA* performs the best. Figure 6.11 shows that for all three move acceptance criteria the CBS approach makes several large improvements early on in the search, but quickly reaches local optima (within ∼1000 iterations). The *OI* run is able to rapidly escape this initial optimum, but gets stuck in a new local optimum, at which it converges prematurely.

Figure 6.11. Change in packing efficiency for the 1BPP Scholl_1 *N3C2W2_N* problem instance, for different move acceptance criteria.



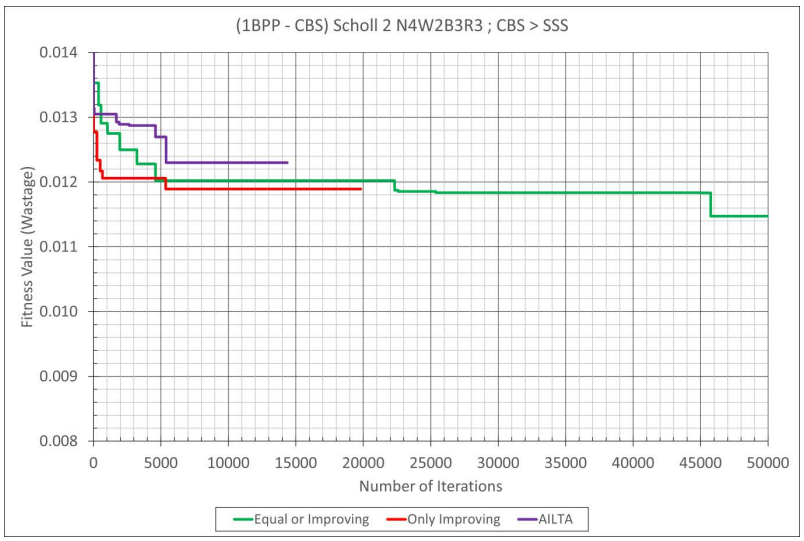
The *EI* run reaches the same initial optimum as that of the *OI* run, but takes considerably longer to escape (∼14000 iterations). In escaping the search reaches a new local optimum, at which it converges prematurely. The final optimum found by the *OI* run is of equally to that found be the *EI* run.

The *AILTA* run initially converges to a local optimum (of equal quality to the final optimum found by the *OI* and *EI* runs), but is able to escape after a considerable amount of time (∼22000 iterations). The escape from the initial optimum is achieved by means of a large improvement to the packing efficiency. This improvement moves the search to a new local optimum, at which it converges prematurely. The search was assumed to have converged, and therefore stopped, after 5000 iterations without any improvements.

Figure 6.12 shows the changes in wastage values for the 1BPP Scholl_2 *N4W2B3R3* problem instance. The figure shows that for this problem instance the run using *EI* move acceptance is the best. From the figure one can see that for the *OI* run the search rapidly makes several large improvements before quickly reaching a local optimum (within ∼1000 iterations). However, the search is able to escape this local optimum after a relatively short period of time (∼5000 iterations) and reach a new local optimum, from which it is unable to escape and at which it converges prematurely.

The *AILTA* run reaches a local optimum almost immediately, but quickly escapes (with ∼2000 iterations) through a series of small consecutive improvements until a new local optimum is reached.

Figure 6.12. Change in packing efficiency for the 1BPP Scholl_2 *N4W2B3R3* problem instance, for different move acceptance criteria.



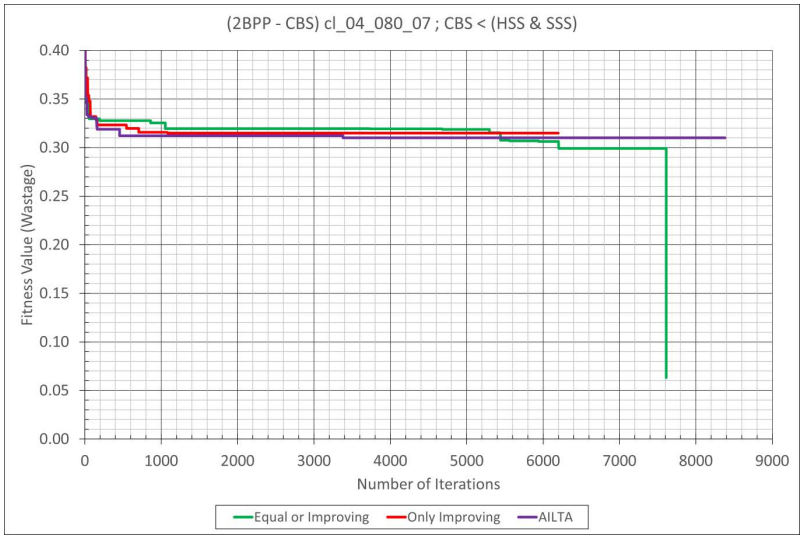
Once again the search is able to quickly escape the local optimum, this time through a moderate improvement followed shortly by a large improvement, leaving the search at a new local optimum where it converges prematurely.

The *EI* run begins with a series of moderate improvements at almost regular intervals before reaching its first local optimum (at ∼5000 iterations), one which is marginally better than the initial optimum for the *OI* run. The *EI* run remains at this optimum for a long period of time (∼17000 iterations) before making a small improvement, thereby escaping to a new local optimum (which is of equal quality to that at which the *OI* run converged). Again the *EI* run remains at the local optimum for a long period of time (∼24000 iterations). A large improvement is then made, moving the search to a new local optimum, where it stays for the remainder of the search. It should be noted that all the optima encountered by the *AILTA* run are of worse quality than any encountered by the *OI* or *EI* runs.

6.3.2.1 Move Acceptance Criteria - 2BPP

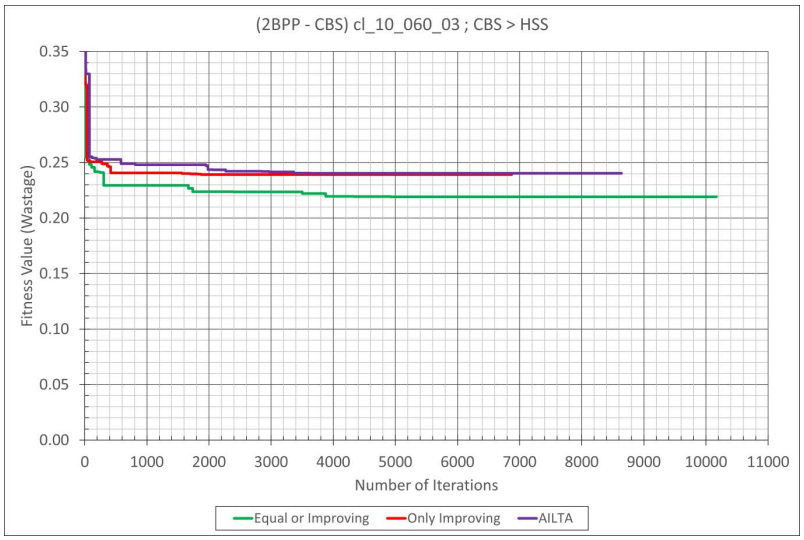
Figure 6.13 shows the change in wastage values for the 2BPP *cl_04_080_07* problem instance, where it can be seen that the *EI* run is the best performing. The figure shows that all three move acceptance criteria follow the same pattern, and have very similar values, up to approximately 6000 iterations. At this point the *OI* run converges prematurely whilst the *AILTA* run continues along its local optimum before also converging prematurely at approximately 8500 iterations. The *EI* run however, makes a

Figure 6.13. Change in packing efficiency for the CBS approach using different move acceptance criteria for the 2BPP *cl_04_080_07* problem instance.



small improvement, briefly moving to a new local optimum (for ∼1400 iterations) followed by a very large improvement, taking the search to the global optimum.

Figure 6.14. Change in packing efficiency for the CBS approach using different move acceptance criteria for the 2BPP *cl_10_060_03* problem instance.



The change in wastage values for the 2BPP *cl_10_060_03* problem instance are shown in Fig. 6.14. The figure shows that *EI* is the best run. As with the previous problem instance (*cl_04_080_07*) all three runs follow a similar pattern. All three make large, rapid improvements to the fitness values within the first 200 iterations. From there *EI* flattens out into a local optimum which it soon escapes. *EI* continues to explore local optima for different lengths of time (iterations) before escaping them.

This happens up until approximately 3500 iterations where *EI* enters a local optimum that it cannot escape from, where it converges prematurely.

Figure 6.15. Change in packing efficiency for the CBS approach using different move acceptance criteria for the 2BPP *cl_01_060_09* problem instance.

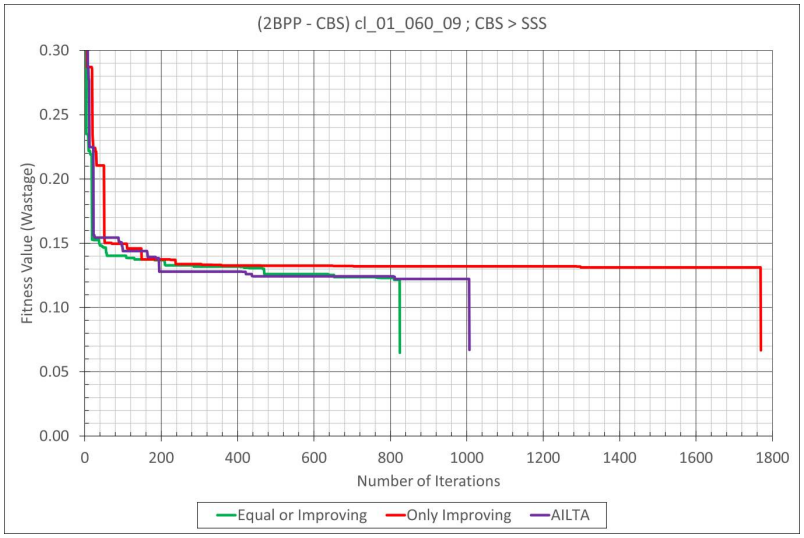
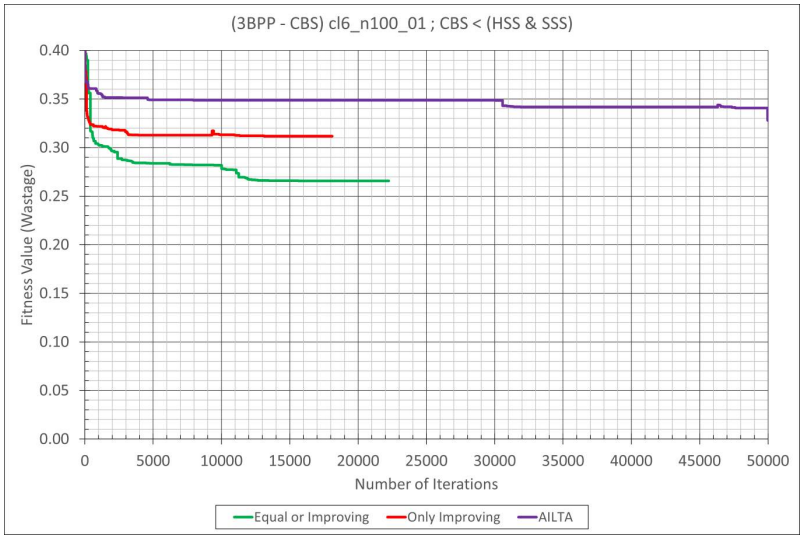


Figure 6.15 shows the change in wastage values for the 2BPP *cl_01_060_09* problem instance. The figure shows all three runs converge to the same value, but the *EI* run does so the quickest. In the figure it can be seen that all three runs follow a similar pattern, making a few large improvements in the early stages of the search (within the first 100 iterations). All three runs continue to make small to moderate improvements over the next 100 iterations before each reaching respective local optimum. After varying periods of time all three runs are able to escape to the global optimum. The *OI* run enters the worst local optimum (out of the three runs) and remains there for approximately 1500 iterations before escaping to the global optimum (at $\sim$1800 iterations). The *AILTA* run enters the best local optimum (out of the three runs) and makes a further small improvement after approximately 200 iterations. The search remains at this new optimum for approximately 600 further iterations before making the final large improvement to reach the global optimum (at $\sim$1000 iterations). The *EI* run enters an initial local optimum of equal quality to that of the *EI* run, but makes a small improvement (after $\sim$250 iterations) taking the search to an area of equal quality to that of the *AILTA* run. The search remains at this optimum for approximately 300 iterations before making the final large improvement to reach the global optimum (at $\sim$800 iterations). Thus, whilst all three runs for this problem instance reach the global optimum, the *EI* run uses the fewest iterations to do so.

6.3.2.2 Move Acceptance Criteria - 3BPP

Figure 6.16. Change in packing efficiency for the CBS approach using different move acceptance criteria for the 3BPP *cl6_n100_01* problem instance.



The changes in wastage value for the 3BPP *cl6_n100_01* problem instance are shown in Fig. 6.16 The figure shows the *EI* run to be the best of the three. The figure illustrates that the *AILTA* run initially makes a moderately large improvement, followed by several smaller improvements, before reaching a local optimum (at ∼1000 iterations). The *AILTA* run remains at this optimum for a long period of time (approximately 30000 iterations) before making a small improvement, moving the search to a new local optimum. The search remains at this optimum until just before the end of the search (for ∼20000 iterations) where it makes a final moderate improvement before terminating (all simulations were conducted imposing a maximum limit of a total of 50000 iterations).

The *OI* run makes a few large improvements within the first ∼500 iterations, continuing with numerous small improvements over the next ∼3500 iterations, at which point the search enters a local optimum. Being unable to escape this local optimum, the *OI* run converges prematurely.

As with the other runs, the *EI* run makes several large improvements within the first ∼500 iterations. These large improvements are then followed by numerous small improvements up until approximately 4000 iterations (with a single moderately sized improvement being made at ∼2500 iterations). At approximately 4000 iterations the search enters its first local optimum, at which it remains for approximately 6000 iterations. A further series of smaller improvements are then made before the search enters a new local optimum at approximately 13000 iterations. The search converges prematurely, being unable to escape this new local optimum.

It is worth noting that after approximately 500 iterations, there is a distinct separation between the fitness curves of the three runs, and that no two fitness curves overlap. The *AILTA* run has the worst fitness value, followed by the *OI* run, with the *EI* run having the best fitness value. This shows that for this problem instance, although using *EI* as a move acceptance criterion leads to premature convergence, it also enables the search to find solutions with less wasted space across the bins (i.e. ones with better fitness values, as the fitness is a measure of the wasted space across the bins in a given solution).

Figure 6.17. Change in packing efficiency for the CBS approach using different move acceptance criteria for the 3BPP *cl6_n200_02* problem instance.

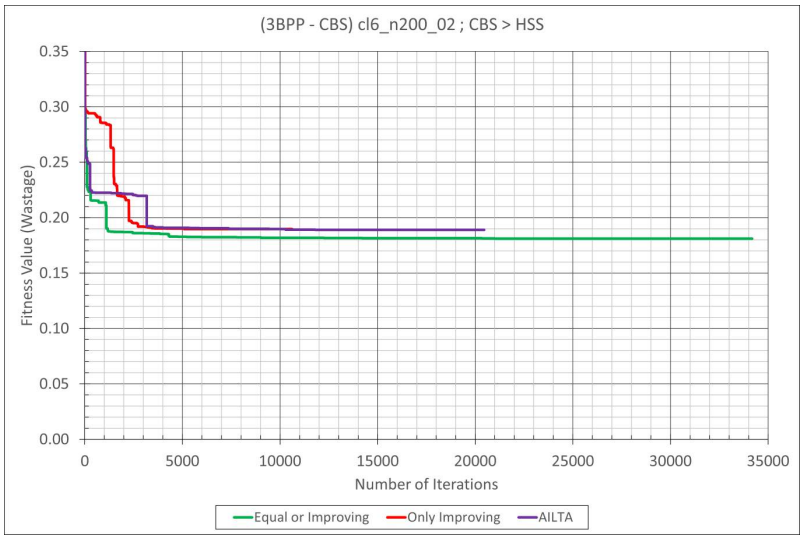


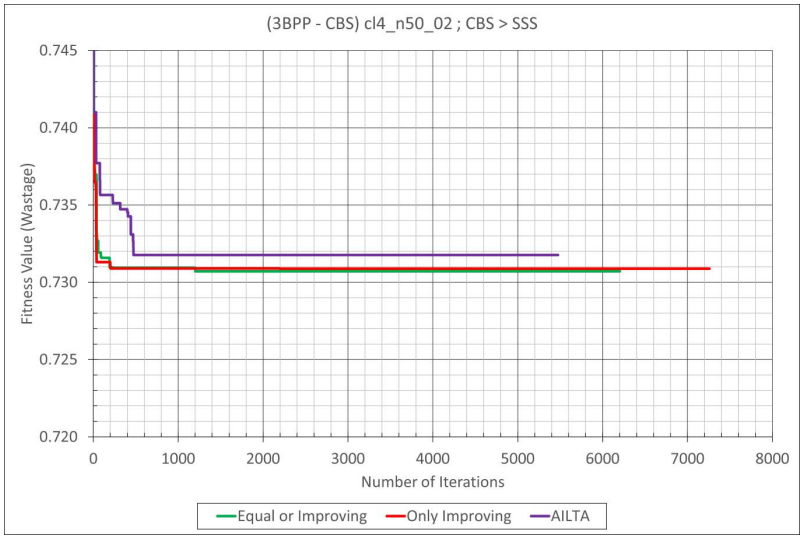
Figure 6.17 shows the change in wastage values for the 3BPP *cl6_n200_02* problem instance. The figure shows the *EI* run to be the best of the three. From the figure one can see that the run for the *AILTA* move acceptance criterion initially makes a large improvement, but quickly reaches a plateau in the fitness values (within 500 iterations). However, the search is able to continue making small improvements around this value until at approximately 3000 iterations a large improvement is made, moving the search to a new local optimum. The *AILTA*-based CBS search is unable to make sufficient further improvements in order to escape the new area of the search space and ultimately the search converges prematurely.

The *EI* run follows a similar pattern to the *AILTA* run, making large improvements early on in the search before reaching a plateau (again, within 500 iterations). However, the first plateau reached by the *EI* run is of better quality (i.e. lower fitness value) than that of the *AILTA* run. Furthermore, the *EI* run escapes the plateau in a much shorter time (within $\sim$1000 iterations as opposed to the $\sim$3000

iterations in the *AILTA* run), and escapes to a local optimum of better quality than that of either of the other two runs.

In contrast to the *EI* and *AILTA* runs, the *OI* run begins with a series of small improvements for the first approximately 1500 iterations, at which point the search makes several large improvements, before reaching a small plateau in the fitness value. The quality of this plateau is of equal quality to that initially encountered by the *AILTA* run. However, the *OI* run rapidly escapes the plateau (within ∼500 iterations) to reach a local optimum of equal quality to that at which the *AILTA* run converged. Similarly, the *OI* run is unable to escape this local optimum and also converges prematurely.

Figure 6.18. Change in packing efficiency for the CBS approach using different move acceptance criteria for the 3BPP *cl4_n50_02* problem instance.



The changes in wastage values for the 3BPP *cl4_n50_02* problem instance are shown in Fig. 6.18, where it can be seen that the run using *EI* move acceptance is the best. From Fig. 6.18 it can be seen that the run using the *AILTA* move acceptance criterion initially makes one or two large improvements, plateaus briefly, and continues making improvements to the fitness value, with the improvements gradually growing in magnitude. At approximately 500 iterations the search reaches a local optimum, from which it is unable to escape and converges prematurely.

The *OI* and *EI* runs follow identical patterns to one another, making very large improvements in the beginning of the search, before reaching a small plateau that is quickly escaped (within ∼200 iterations). Each of the *OI* and *EI* runs escape this initial plateau into local optima of differing quality, with the *EI* run reaching a point of marginally better quality than that of the *OI* run. Both the *OI* and *EI*

runs converge prematurely, with the searches being unable to escape their new respective local optima. It is worth noting that the *AILTA* run reaches a local optimum of notably worse quality than that of either the *OI* or *EI* runs.

6.3.3 Discussion

For 1BPP the fitness curves for the different move acceptance criteria (Figs. 6.10, 6.11 and 6.12) showed that whilst the *EI* criterion tends to be the best choice when applying CBS (as in the case of the Falkenauer_T *t501_19* and Scholl_2 *N4W2B3R3* 1BPP problem instances), it is not always the best choice, as evidenced by the Scholl_1 *N3C2W2_N* 1BPP problem instance. The deviation from the general trend of *EI* being the best acceptance criterion for 1BPP problem instances indicates that the CBS approach is sensitive, to some degree, to the choice of move acceptance criterion.

The trend of *EI* being the most effective choice of acceptance criterion for CBS is again seen in the move acceptance fitness plots for the 2BPP problem instances (Figs. 6.13, 6.14 and 6.15). Although the shape of the fitness curves differ for each problem instance, the three different move acceptance runs initially follow the same trend (for the same problem instance). For the *cl_04_080_07* problem instance the *EI* run is able to find the global optimum, whilst the two other runs merely converge prematurely. On the *cl_01_060_09* problem instance all three runs found the global optimum, however the *EI* run did so in the least number of iterations. The third problem instance (*cl_10_060_03*) showed that, although all three runs converged prematurely, the *EI* run took longer to converge and found better quality solutions during the search than either of the two other runs. At no point in the search is the fitness curve of the *EI* run worse than either of the two other runs.

As with the other problem dimensions, the 3BPP move acceptance fitness curves (Figs. 6.16, 6.17 and 6.18) indicate the greater effectiveness of using the *EI* acceptance criterion over either the *OI* or the *AILTA* criteria. Problem instances *cl6_n100_01* and *cl6_n200_02* showed that, in spite of converging prematurely, the *EI* run found solutions of a considerably greater quality than the other two runs. As with the 2BPP *cl_10_060_03* problem instance, for both problem instances, there is no point at which the *EI* run has a worse fitness value than either of the two other runs. The fitness curves for problem instance *cl4_n50_02* (Fig. 6.18) show a negligible difference between the *EI* and *OI* runs, however both runs have considerably better fitness values than the *AILTA* run.

In conclusion, when considering the effectiveness of the three different move acceptance criteria (*EI*, *OI* and *AILTA*) for the CBS approach, across all three problem domains (1BPP, 2BPP and 3BPP), one can see a trend of *EI* being the most effective choice for CBS. However, there are instances where *EI* is not the best choice (e.g. the 1BPP Scholl_1 *N3C2W2_N* problem instance). It is interesting to note that the somewhat sophisticated *AILTA* criterion is generally the worst one to use. The fact that the performance of the CBS approach differs according to the choice of move acceptance criterion indicates that the approach is sensitive to the move acceptance criterion used.

6.4 COMPARISON WITH PREVIOUS BI-SPACE APPROACHES

This section presents a comparison of the CBS approach with the previous bi-space search approaches BSS and CSA (discussed in Section 2.4). First a comparison of bi-space search approaches is made using the number of bins. This is followed by a comparison according to average runtimes per problem instance. The comparisons are only made for 1BPP as neither BSS nor CSA were implemented for 2BPP or 3BPP.

Table 6.10 presents the number of 1BPP problem instances solved to optimum or near-optimum for the CBS approach and the previous bi-space search approaches BSS [8] and CSA [7]. From the table it can be seen that CBS outperforms both the CSA and BSS approaches for all three *Scholl* datasets. In addition, CBS outperforms BSS for the *Falkenauer* datasets and has equivalent performance to BSS for the *Hard28* dataset. Although CBS does not outperform BSS for the *Schwerin* and *Waescher* datasets, the difference in performance is small, being at most three problem instances.

Table 6.10. Number of problem instances solved to optimum by bi-space search approaches. The best results are reported in bold.

	CBS			BSS [8]			CSA [7]		
	Opt.	Opt. - 1	Sum	Opt.	Opt. - 1	Sum	Opt.	Opt. - 1	Sum
Falkenauer_T	**1**	**79**	**80**	0	9	9	-	-	-
Falkenauer_U	**50**	**30**	**80**	35	37	72	-	-	-
Scholl_1	**695**	**25**	**720**	671	44	715	673	44	717
Scholl_2	**463**	**16**	**479**	408	41	449	428	40	468
Scholl_3	**8**	**2**	**10**	5	5	10	6	4	10
Schwerin_1	97	3	100	**100**	**0**	**100**	-	-	-
Schwerin_2	98	2	100	**100**	**0**	**100**	-	-	-
Hard28	**5**	**23**	**28**	**5**	**23**	**28**	-	-	-
Waescher	**12**	**5**	**17**	**12**	**5**	**17**	-	-	-

The average runtime (in seconds) per problem instance for each 1BPP dataset is presented in Table 6.11. From the table it can be seen that CBS has the lowest average runtime for all datasets, except for the *Schwerin_1* dataset for which it is approximately half a second slower than BSS. As was pointed out by the authors Beckedahl and Pillay [8], CSA may be an improvement over BSS, but the increase in runtimes relative to the increase in performance makes CSA an impractical choice over BSS. In contrast, however, CBS not only has improved performance over both BSS and CSA (as indicated in Table 6.10), but the approach also has the lowest average runtimes for all datasets, making it the most practical bi-space search approach for solving 1BPP instances.

Table 6.11. Average runtime (in seconds) per 1BPP problem instance for each dataset. The best results are reported in bold.

Problem Set	CBS	BSS [8]	CSA [7]
Falkenauer_T	**8.54**	22.28	-
Falkenauer_U	**17.75**	81.87	-
Scholl_1	**4.45**	26.58	817.50
Scholl_2	**3.26**	5.44	1589.80
Scholl_3	**8.89**	14.34	1761.10
Schwerin_1	0.74	**0.22**	-
Schwerin_2	**1.29**	1.86	-
Hard28	**5.75**	16.34	-
Waescher	**2.32**	5.43	-

6.5 COMPARISON WITH STATE-OF-THE-ART

This section presents an empirical comparison of the CBS approach to the state-of-the-art approaches taken from the literature. The comparisons are made using the number of bins in the solutions found by the searches. The comparisons are made for each of the 1BPP, 2BPP and 3BPP problem domains, in order.

Table 6.12 reports the number of 1BPP problem instances that were solved to the optimum number of bins for the state-of-the-art solution space searches and hyper-heuristics, namely Perturbation-SAWMBS (P.-SAWMBS) [25], grouping genetic algorithm with controlled gene transmission (GGA-CGT) [26], consistent neighbourhood search for bin packing (CNS_BP) [27] and simulated annealing hyper-heuristic (SAHH) [36] and the CBS approach. These approaches are described in detail in Sections 3.1.1 and 3.1.2. From Table 6.12 it can be seen that the CBS approach outperforms the SAHH hyper-heuristic approach for the Scholl_3 dataset but does not outperform any of the state-of-the-art

Table 6.12. Number of 1BPP problem instances solved to optimum. The best results are reported in bold.

Problem Set	P.-SAWMBS	GGA-CGT	CNS_BP	SAHH	CBS
Falkenauer_T (80)	79	**80**	**80**	76.5	1
Falkenauer_U (80)	**80**	**80**	**80**	79.8	50
Scholl_1 (720)	**720**	**720**	**720**	697.5	695
Scholl_2 (480)	**480**	**480**	**480**	473.4	463
Scholl_3 (10)	**10**	**10**	**10**	7.3	8
Schwerin_1 (100)	**100**	**100**	**100**	-	97
Schwerin_2 (100)	**100**	**100**	**100**	-	98
Hard28 (28)	5	16	**25**	-	5
Wäscher (17)	16	16	**17**	-	12

approaches for any of the remaining datasets. Future work will investigate why this is the case as for 2BPP and 3BPP there are cases where CBS has equal or better performance than state-of-the-art.

Tables 6.13 and 6.13 reports the average number of bins found by CBS and the state-of-the-art for 2BPP problem instance classes 1-5 and 6-10 respectively. As was done previously, and in order to remain consistent with the literature, the reported values are the average number of bins across the ten instances for each subclass of problem instance. The state-of-the-art approaches are the Biased Random Key Genetic Algorithm (BRKGA) [58], the GRASP Variable Neighbourhood Descent (GVND) [57], the Set-Covering-Based Heuristic (SCH) [53], the Guided Local Search (GLS) [54], Tabu Search (TS2) [51], a heuristic approach (HBP) [52] and a Hyper-heuristic Algorithm for the Oriented 2BPP (HHA-O) [60]. These approaches are described in detail in Chapter 2.

From the tables one can see that CBS is the best performing or tied for best performing approach for problem instance classes 1,2 and 9. Of the remaining subclasses, CBS is the second best performing for 7, being only 0.1 bins away from the best approach. That leaves a total of 3 subclasses for which CBS performs worse than second. The majority of the subclasses for which CBS is not best performing are those for which n = 40 or n = 100. Future work will investigate why these types of problem instances present a challenge for CBS.

The CBS approach is compared with state-of-the-art techniques for each of the 1BPP, 2BPP and 3BPP problem domains. For the simpler 1BPP domain the CBS approach is not found to outperform the state-of-the-art techniques. However, for the more difficult 2BPP and 3BPP domains the CBS approach

Table 6.13. Number of bins, averaged across ten instances, obtained for 2BPP problem instance classes 1-5. The best results are reported in bold.

Class	Bin size	n	LB	BRKGA	GVND	SCH	GLS	TS3	HBP	HHA-O	CBS
1	10x10	20	7.1	**7.1**	**7.1**	**7.1**	**7.1**	**7.1**	**7.1**	**7.1**	**7.1**
		40	13.4	**13.4**	**13.4**	**13.4**	**13.4**	13.5	**13.4**	13.6	**13.4**
		60	19.7	**20.0**	**20.0**	**20.0**	20.1	20.1	20.1	20.2	**20.0**
		80	27.4	27.5	27.5	27.5	27.5	28.2	27.5	27.7	**27.4**
		100	31.7	**31.7**	**31.7**	**31.7**	32.1	32.6	31.8	32.3	**31.7**
2	30x30	20	1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
		40	1.9	**1.9**	**1.9**	**1.9**	**1.9**	2.0	**1.9**	2.0	**1.9**
		60	2.5	**2.5**	**2.5**	**2.5**	**2.5**	2.7	**2.5**	2.6	**2.5**
		80	3.1	**3.1**	**3.1**	**3.1**	**3.1**	3.3	**3.1**	3.2	**3.1**
		100	3.9	**3.9**	**3.9**	**3.9**	**3.9**	4.0	**3.9**	4.0	**3.9**
3	40x40	20	5.1	**5.1**	**5.1**	**5.1**	**5.1**	5.5	**5.1**	5.4	**5.1**
		40	9.2	**9.4**	**9.4**	**9.4**	**9.4**	9.7	9.5	9.7	**9.4**
		60	13.6	**13.9**	**13.9**	**13.9**	14.0	14.0	14.0	14.4	**13.9**
		80	18.7	**18.9**	**18.9**	**18.9**	19.1	19.8	19.1	19.8	19.0
		100	22.1	**22.3**	**22.3**	**22.3**	22.6	23.6	22.6	23.3	**22.3**
4	100x100	20	1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
		40	1.9	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**
		60	2.3	**2.5**	**2.5**	**2.5**	**2.5**	2.6	**2.5**	2.7	**2.5**
		80	3.0	**3.1**	**3.1**	3.2	3.3	3.3	3.3	3.4	**3.1**
		100	3.7	**3.7**	3.8	3.8	3.8	4.0	3.8	3.9	3.8
5	100x100	20	6.5	**6.5**	**6.5**	**6.5**	**6.5**	6.6	**6.5**	6.6	**6.5**
		40	11.9	**11.9**	**11.9**	**11.9**	**11.9**	**11.9**	**11.9**	12.4	**11.9**
		60	17.9	**18.0**	**18.0**	**18.0**	18.1	18.2	**18.0**	18.6	**18.0**
		80	24.1	**24.7**	**24.7**	**24.7**	24.9	25.1	24.8	25.3	**24.7**
		100	27.9	**28.1**	28.2	28.2	28.8	29.5	28.7	29.4	28.3

is found to be scalable, having comparable performance to the state-of-the-art techniques and in some cases outperforming them.

Table 6.14. Number of bins, averaged across ten instances, obtained for 2BPP problem instance classes 6-10. The best results are reported in bold.

Class	Bin size	n	LB	BRKGA	GVND	SCH	GLS	TS3	HBP	HHA-O	CBS
6	300x300	20	1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
		40	1.5	**1.6**	1.7	1.7	1.8	1.9	1.8	1.9	1.7
		60	2.1	**2.1**	**2.1**	**2.1**	2.2	2.2	**2.1**	2.3	**2.1**
		80	3.0	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**
		100	3.2	**3.3**	3.4	3.4	3.4	3.4	3.4	3.5	3.4
7	100x100	20	5.5	**5.5**	**5.5**	**5.5**	**5.5**	**5.5**	**5.5**	5.7	**5.5**
		40	10.9	**11.1**	**11.1**	**11.1**	11.3	11.4	**11.1**	11.5	11.2
		60	15.6	**15.8**	15.9	**15.8**	15.9	16.2	16.0	16.1	**15.8**
		80	22.4	**23.2**	**23.2**	**23.2**	**23.2**	**23.2**	**23.2**	**23.2**	**23.2**
		100	26.9	**27.1**	**27.1**	**27.1**	27.5	27.7	27.4	27.6	**27.1**
8	100x100	20	5.8	**5.8**	**5.8**	**5.8**	**5.8**	**5.8**	**5.8**	5.9	**5.8**
		40	11.2	**11.3**	**11.3**	**11.3**	11.4	11.4	**11.3**	11.5	11.4
		60	15.9	**16.1**	**16.1**	16.2	16.3	16.2	16.2	16.6	**16.1**
		80	22.3	22.4	22.4	22.4	22.5	22.6	22.6	22.7	**22.3**
		100	27.4	27.8	27.8	27.9	28.1	28.4	28.0	28.4	**27.7**
9	100x100	20	14.3	**14.3**	**14.3**	**14.3**	**14.3**	**14.3**	**14.3**	**14.3**	**14.3**
		40	27.8	**27.8**	**27.8**	**27.8**	**27.8**	**27.8**	**27.8**	**27.8**	**27.8**
		60	43.7	**43.7**	**43.7**	**43.7**	**43.7**	43.8	**43.7**	**43.7**	**43.7**
		80	57.7	**57.7**	**57.7**	**57.7**	**57.7**	**57.7**	**57.7**	**57.7**	**57.7**
		100	69.5	**69.5**	**69.5**	**69.5**	**69.5**	**69.5**	**69.5**	**69.5**	**69.5**
10	100x100	20	4.2	**4.2**	**4.2**	**4.2**	**4.2**	4.3	4.3	4.3	**4.2**
		40	7.4	**7.4**	**7.4**	**7.4**	**7.4**	7.5	**7.4**	7.6	**7.4**
		60	9.8	**10.0**	**10.0**	10.1	10.2	10.4	10.2	10.6	10.3
		80	12.3	**12.8**	12.9	**12.8**	13.0	13.0	13.0	13.4	13.0
		100	15.3	**15.8**	15.9	15.9	16.2	16.6	16.2	16.4	15.9

Table 6.15. Ratio of number of bins to the lower bound for 3BPP. The best results are reported in bold.

Class	Bin Size	n	L2	BRKGA	TS3	GVND	TS2P	GLS	CBS
1	100x100x100	50	12.5	1.072	1.072	1.072	1.072	1.072	**1.036**
		100	23.2	1.060	1.060	1.060	1.064	1.064	**1.056**
		150	37.2	**1.049**	1.058	**1.049**	1.066	1.066	1.050
		200	47.6	1.050	1.058	1.052	1.056	1.058	**1.042**
2	100x100x100	50	12.4	1.087	1.087	1.087	-	-	**1.040**
		100	24.3	**1.062**	1.066	1.066	-	-	**1.062**
		150	36.0	**1.043**	1.060	1.051	-	-	1.062
		200	47.2	1.040	1.055	1.040	-	-	**1.034**
3	100x100x100	50	12.5	1.081	1.081	1.081	-	-	**1.033**
		100	23.1	**1.049**	1.053	1.053	-	-	1.085
		150	34.8	**1.042**	1.047	1.044	-	-	1.047
		200	48.8	**1.042**	1.056	1.046	-	-	1.048
4	100x100x100	50	31.2	1.024	1.024	1.024	1.024	1.024	**1.000**
		100	57.0	1.024	1.024	1.024	**1.023**	1.024	1.026
		150	87.4	**1.019**	**1.019**	**1.019**	**1.019**	**1.019**	1.023
		200	113.5	**1.021**	**1.021**	**1.021**	**1.021**	1.023	1.024
5	100x100x100	50	7.9	1.137	1.151	1.137	1.137	1.137	**1.062**
		100	12.0	**1.163**	**1.163**	**1.163**	1.178	1.171	1.170
		150	16.0	**1.149**	1.172	1.155	1.155	1.161	**1.149**
		200	22.0	**1.111**	1.131	**1.111**	1.123	1.115	1.143
6	10x10x10	50	10.5	1.115	1.138	1.126	1.126	1.126	**1.000**
		100	18.0	1.080	1.091	1.086	1.091	1.091	**1.078**
		150	26.7	**1.078**	1.093	1.086	1.086	1.093	1.094
		200	36.3	**1.066**	1.077	1.069	1.077	1.077	1.075
7	40x40x40	50	5.7	1.175	1.190	1.175	1.175	1.175	**1.144**
		100	12.1	**1.119**	1.147	1.147	1.128	1.128	1.162
		150	15.7	**1.117**	1.175	1.168	1.153	1.153	1.132
		200	20.4	**1.114**	1.138	1.119	1.119	1.119	1.158
8	100x100x100	50	9.4	1.150	1.163	1.150	1.150	1.150	**1.013**
		100	17.2	1.080	1.080	1.080	**1.074**	1.080	1.079
		150	20.4	**1.108**	1.131	1.131	1.122	1.122	1.131
		200	27.1	**1.097**	1.135	1.116	1.124	1.120	1.164

CHAPTER 7 CONCLUSIONS AND FUTURE WORK

This chapter provides a summary of the work presented in this thesis, and discusses the main findings and contributions. A discussion for each of the objectives outlined in Section 1.2 of Chapter 1 is presented in Section 7.1. A summary of the overall findings is also presented. Finally, future work is discussed in Section 7.2.

7.1 SUMMARY AND DISCUSSION

The conclusions drawn for each of the objectives outlined in Section 1.2 of Chapter 1 are presented below, followed by a summary.

1. **To develop an algorithm that searches the solution space to solve bin packing problems:** To achieve this objective, an algorithm, SSS (Solution Space Search), was developed that used an LS to explore the solution space for bin packing problems. A separate algorithm was developed for 1BPP, 2BPP and 3BPP. The LS was chosen over more complex optimisation techniques, such as evolutionary algorithms, due to its lower computational cost because the objective of this study was not to optimise the performance of the algorithm, but rather to establish the effectiveness of concurrently searching two search spaces.
2. **To develop an algorithm that searches the heuristic space to solve bin packing problems.** This objective was met by developing an algorithm, HSS (Heuristic Space Search), that used an LS to search through the heuristic space of bin packing problems. The same LS that was used for the solution space search was used for the heuristic space search with the same rationale behind the choice of using an LS. A separate algorithm was developed for 1BPP, 2BPP and 3BPP. As part of the design process, a set of low-level heuristics needed to be selected for each of the

bin packing problems. These sets of low-level heuristics were created by selecting the most commonly used low-level heuristics from the literature for each of the bin packing problems.

3. **To develop a CBS algorithm that searches the heuristic and solution spaces concurrently.**
   To meet this objective, a CBS algorithm was developed that used an LS to concurrently search the heuristic and solution spaces of bin packing problems. The same LS as was used for the single space searches was used for the CBS for the same reason.
4. **To compare the effectiveness of CBS to that of solution space search.**
   This objective was achieved by applying both the solution space search algorithm and the CBS algorithm to a large benchmark set of problem instances for 1BPP, 2BPP and 3BPP. For each problem instance, the CBS was compared to SSS in terms of the total number of bins used by the solution found, the fitness value (which is a measure of the total wasted space across all bins) and the computational time taken to find a solution. For the majority of problem instances for 1BPP, 2BPP and 3BPP, CBS found solutions that used fewer bins and had lower fitness values than SSS. SSS had lower computational times than CBS, but the computational times of CBS were still low enough to be considered practical.
5. **To compare the effectiveness of CBS to that of heuristic space search.**
   To meet this objective, both CBS and HSS were applied to 1BPP, 2BPP and 3BPP. For each problem instance, the two algorithms were compared to one another in terms of the total number of bins used by the solution found, the fitness value and the computational time taken to find a solution. For 1BPP, 2BPP and 3BPP, CBS found solutions that used fewer bins and had lower fitness values than HSS for the majority of problem instances. CBS also had lower computational times than HSS.

This study proposed an approach that concurrently searches the heuristic and solution spaces (i.e. Concurrent Bi-space Search, CBS). The effectiveness of the approach was evaluated using 1BPP, 2BPP and 3BPP, and its performance was compared to that of single-space searches. Both the single-space searches (SSS and HSS) and the CBS used the same LS to search. Experimental results showed that CBS outperformed both SSS and HSS in terms of the number of bins used and the fitness value for the majority of problem instances for 1BPP, 2BPP and 3BPP. However, SSS had lower computational times than CBS, with CBS having lower computational times than HSS. In spite of CBS having higher computational times than SSS, the runtimes were still found to be within practical limits. For 2BPP and 3BPP, the CBS approach was found to be scalable to more complex problems. Additionally, the CBS approach was found to be competitive with the state-of-the-art techniques for 2BPP and 3BPP.

7.2 FUTURE WORK

Future extensions to the research presented in this study will involve the following:

- **Other optimisation techniques:**

  This study used an LS for the CBS algorithm due to its lower computational time. However, this came at the expense of the quality of the solutions found. Future work will investigate using other approaches, to concurrently search the two spaces in an effort to optimise the performance of the CBS approach.

- **Use of Exploratory Landscape Analysis (ELA) features:**

  Exploratory Landscape Analysis (ELA) is a technique that can be used to analyse the structure or landscape of a given search space. Future work will involve using ELA to analyse the landscape of the heuristic and solution spaces during the search process, in an effort to gain a better understanding of the neighbourhood structure of the two spaces as part of the concurrent space search. This information could then be incorporated into the CBS used to search the two spaces.

- **Extension to other/additional search spaces:**

  Whilst this study has focused specifically on concurrently searching the heuristic and solution spaces, there exist many other search spaces such as the design space in automated design or the program space in genetic programming. Future work will involve extending the CBS approach to search in these other spaces, as well as extending the idea of bi-space search to that of multi-space search (more than two search spaces).

- **Extension to other problem domains:**

  Although this study has been restricted to bin packing problems, the CBS approach is not restricted to this domains. Hence, future work will investigate the effectiveness of the CBS approach on other optimisation problems, both combinatorial and continuous.

REFERENCES

1. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Publishing, 2016. ISBN 9781537600314.

2. T. Gonzalez. *Handbook of Approximation Algorithms and Metaheuristics: Methodologies and Traditional Applications, Volume 1*. Chapman & Hall/CRC Computer and Information Science Series. CRC Press, 2018. ISBN 9781351236409.

3. J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. A Bradford book. Bradford, 1992. ISBN 9780262111706.

4. N. Pillay and R. Qu. *Hyper-Heuristics: Theory and Applications*. Natural Computing Series. Springer International Publishing, 2018. ISBN 978-3-319-96514-7.

5. E. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, December 2013. doi: 10.1057/jors.2013.71.

6. R. Qu and E. Burke. Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society*, 60(9):1273–1285, 2009. doi: 10.1057/jors.2008.102.

7. D. Beckedahl and N. Pillay. A study of bi-space search for solving the one-dimensional bin packing problem. In L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz,

and J. Zurada, editors, *Artificial Intelligence and Soft Computing*, pages 277–289, Cham, 2020. Springer International Publishing. ISBN 978-3-030-61534-5.

8. D. Beckedahl and N. Pillay. Bi-space search: Optimizing the hybridization of search spaces in solving the one dimensional bin packing problem. In L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz, and J. Zurada, editors, *Artificial Intelligence and Soft Computing*, pages 206–217, Cham, 2023. Springer International Publishing. ISBN 978-3-031-23480-4.

9. B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 2018. ISBN 9783662560396.

10. G. Scheithauer. *Introduction to Cutting and Packing Optimization: Problems, Modeling Approaches, Solution Methods*. Springer International Publishing, Cham, 2018. ISBN 978-3-319-64403-5.

11. R. Alvarez-Valdes, M. Carravilla, and J. Oliveira. *Cutting and Packing*, pages 931–977. Springer International Publishing, Cham, 2018. ISBN 978-3-319-07124-4. doi: 10.1007/978-3-319-07124-4_43.

12. H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990. ISSN 0377-2217. doi: 10.1016/0377-2217(90)90350-K.

13. A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11(4):345–357, 1999. doi: 10.1287/ijoc.11.4.345.

14. G. Wäscher, H. Haubner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007. ISSN 0377-2217. doi: 10.1016/j.ejor.2005.12.047.

15. M. Delorme, M. Iori, and S. Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20, 2016. ISSN 0377-2217. doi: 10.1016/j.ejor.2016.04.030.

16. H. Terashima-Marín, P. Ross, C. Farías-Zárate, E. López-Camacho, and M. Valenzuela-Rendón. Generalized hyper-heuristics for solving 2d regular and irregular packing problems. *Annals of Operations Research*, 179(1):369–392, Sep 2010. ISSN 1572-9338. doi: 10.1007/s10479-008-0475-2.

17. S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28(1):59–70, 1990. ISSN 0166-218X. doi: 10.1016/0166-218X(90)90094-S.

18. A. Scholl, R. Klein, and C. Jürgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7):627–645, July 1997. ISSN 0305-0548.

19. F. Glover. Tabu search—part i. *ORSA Journal on Computing*, 1(3):190–206, 1989. doi: 10.1287/ijoc.1.3.190.

20. E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2(1): 5–30, June 1996. ISSN 1572-9397.

21. E. Falkenauer. The grouping genetic algorithms: widening the scope of the ga's. *JORBEL-Belgian Journal of Operations Research, Statistics, and Computer Science*, 33(1-2):79–102, 1993.

22. A. Alvim, C. C. Ribeiro, F. Glover, and D. Aloise. A hybrid improvement heuristic for the one-dimensional bin packing problem. *Journal of Heuristics*, 10(2):205–229, March 2004. doi: 10.1023/b:heur.0000026267.44673.ed.

23. K. Fleszar and K. Hindi. New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 29(7):821–839, 2002. ISSN 0305-0548. doi: 10.1016/S0305-0548(00)00082-4.

24. J. Gupta and J. Ho. A new heuristic algorithm for the one-dimensional bin-packing problem. *Production Planning & Control*, 10(6):598–603, January 1999. doi: 10.1080/095372899232894.

25. K. Fleszar and C. Charalambous. Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem. *European Journal of Operational Research*, 210(2):176–184, 2011. ISSN 0377-2217. doi: 10.1016/j.ejor.2010.11.004.

26. M. Quiroz-Castellanos, L. Cruz-Reyes, J. Torres-Jimenez, C. Gómez, J. Héctor, and A. Alvim. A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Computers & Operations Research*, 55:52–64, 2015. ISSN 0305-0548.

27. M. Buljubašić and M. Vasquez. Consistent neighborhood search for one-dimensional bin packing and two-dimensional vector packing. *Computers & Operations Research*, 76:12–21, December 2016. ISSN 0305-0548. doi: 10.1016/j.cor.2016.06.009.

28. E. G. Coffman, M. R. Garey, and D. S. Johnson. *Approximation Algorithms for Bin-Packing — An Updated Survey*, pages 49–106. Springer Vienna, Vienna, 1984. ISBN 978-3-7091-4338-4. doi: 10.1007/978-3-7091-4338-4_3.

29. D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley series in artificial intelligence. Addison-Wesley, 1989. ISBN 9780201157673.

30. P. Ross, S. Schulenburg, J. Marín-Bläzquez, and E. Hart. Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, GECCO'02, pages 942–948, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1558608788.

31. S. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, June 1995. ISSN 1063-6560. doi: 10.1162/evco.1995.3.2.149.

32. P. Ross, J. Marín-Blázquez, S. Schulenburg, and E. Hart. Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics. In E. Cantú-Paz, J. Foster, K. Deb, L. Davis, R. Roy, U. O'Reilly, H. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. Potter, A. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation — GECCO 2003*, pages 1295–1306, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45110-5. doi: 10.1007/3-540-45110-2_5.

33. K. Sim, E. Hart, and B. Paechter. A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution. In C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, pages 348–357, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-32964-7. doi: 10.1007/978-3-642-32964-7_35.

34. E. Burke, M. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In T. Runarsson, H. Beyer, E. Burke, J. Merelo-Guervós, L. Whitley, and X. Yao, editors, *Parallel Problem Solving from Nature - PPSN IX*, pages 860–869, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-38991-0. doi: 10.1007/11844297_87.

35. P. van Laarhoven and E. Aarts. *Simulated annealing*. Springer Netherlands, Dordrecht, 1987. ISBN 978-94-015-7744-1. doi: 10.1007/978-94-015-7744-1_2.

36. R. Bai, J. Blazewicz, E. Burke, G. Kendall, and B. McCollum. A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR*, 10(1):43–66, March 2012. ISSN 1614-2411. doi: 10.1007/s10288-011-0182-8.

37. K. Sim and E. Hart. Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1549–1556, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450319638. doi: 10.1145/2463372.2463555.

38. D. Jackson. Single node genetic programming on problems with side effects. In C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, pages 327–336, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-32937-1.

39. E. Burke, M. Hyde, and G. Kendall. Grammatical evolution of local search heuristics. *IEEE Transactions on Evolutionary Computation*, 16(3):406–417, 2012. doi: 10.1109/TEVC.2011.2160401.

40. C. Ryan, M. O'Neill, and J. Collins. *Handbook of Grammatical Evolution*. Springer International Publishing, 2018. ISBN 9783319787176.

41. E. López-Camacho, H. Terashima-Marín, and P. Ross. A hyper-heuristic for solving one and two-dimensional bin packing problems. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '11, pages 257–258, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306904. doi: 10.1145/2001858.2002003.

42. E. Burke, M. Hyde, G. Kendall, and J. Woodward. Automating the Packing Heuristic Design Process with Genetic Programming. *Evolutionary Computation*, 20(1):63–89, 03 2012. ISSN 1063-6560. doi: 10.1162/EVCO_a_00044.

43. B. Bengtsson. Packing Rectangular Pieces—A Heuristic Approach. *The Computer Journal*, 25 (3):353–357, 08 1982. ISSN 0010-4620. doi: 10.1093/comjnl/25.3.353.

44. J. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33(1):49–64, 1985. doi: 10.1287/opre.33.1.49.

45. J. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36(4):297–306, 1985. doi: 10.1057/jors.1985.51.

46. N. Christofides and C. Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25(1):30–44, 1977. doi: 10.1287/opre.25.1.30.

47. E. Bischoff and M. Ratcliff. Issues in the development of approaches to container loading. *Omega*, 23(4):377–390, 1995. ISSN 0305-0483. doi: 10.1016/0305-0483(95)00015-G.

48. S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399, March 1998. doi: 10.1287/mnsc.44.3.388.

49. L. Mitten. Branch-and-bound methods: General formulation and properties. *Operations Research*, 18(1):24–34, 1970. doi: 10.1287/opre.18.1.24.

50. J. O. Berkey and P. Y. Wang. Two-dimensional finite bin-packing algorithms. *Journal of the Operational Research Society*, 38(5):423–429, May 1987. doi: 10.1057/jors.1987.70.

51. A. Lodi, S. Martello, and D. Vigo. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, 112(1):158–166, 1999. ISSN 0377-2217. doi: 10.1016/S0377-2217(97)00388-3.

52. M. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. part ii: New lower and upper bounds. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(2):135–147, June 2003. ISSN 1619-4500. doi: 10.1007/s10288-002-0006-y.

53. M. Monaci and P. Toth. A set-covering-based heuristic approach for bin-packing problems. *INFORMS Journal on Computing*, 18(1):71–85, February 2006. doi: 10.1287/ijoc.1040.0089.

54. O. Faroe, D. Pisinger, and M. Zachariasen. Guided local search for the three-dimensional bin-packing problem. *INFORMS Journal on Computing*, 15(3):267–283, August 2003. doi: 10.1287/ijoc.15.3.267.16080.

55. S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, April 2000. doi: 10.1287/opre.48.2.256.12386.

56. T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, Mar 1995. ISSN 1573-2916. doi: 10.1007/BF01096763.

57. F. Parreño, R. Alvarez-Valdes, J. Oliveira, and J. Tamarit. A hybrid GRASP/VND algorithm for two- and three-dimensional bin packing. *Annals of Operations Research*, 179(1):203–220, October 2008. doi: 10.1007/s10479-008-0449-4.

58. J. Gonçalves and M. Resende. A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, 145(2):500–510, October 2013. doi: 10.1016/j.ijpe.2013.04.019.

59. H. Terashima-Marín, E. Flores-Álvarez, and P. Ross. Hyper-heuristics and classifier systems for solving 2d-regular cutting stock problems. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 637–643, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930108. doi: 10.1145/1068009.1068115.

60. M. Beyaz, T. Dokeroglu, and A. Cosar. Robust hyper-heuristic algorithms for the offline oriented/non-oriented 2d bin packing problems. *Applied Soft Computing*, 36:236–245, 2015. ISSN 1568-4946. doi: 10.1016/j.asoc.2015.06.063.

61. H. Terashima-Marin, C. Farias Zarate, P. Ross, and M. Valenzuela-Rendon. Comparing two models to generate hyper-heuristics for the 2d-regular bin-packing problem. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pages 2182–2189, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595936974. doi: 10.1145/1276958.1277377.

62. J.E. Beasley. Operations research library. *Collection of problems for 2D packing and cutting*, 2003. URL https://www.brunel.ac.uk/~mastjjb/jeb/info.html.

63. A. Lodi, S. Martello, and D. Vigo. Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research*, 141(2):410–420, 2002. ISSN 0377-2217. doi: 10.1016/S0377-2217(02)00134-0.

64. T. Crainic, G. Perboli, and R. Tadei. Ts2pack: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research*, 195(3):744–760, 2009. ISSN 0377-2217. doi: 10.1016/j.ejor.2007.06.063.

65. T. Crainic, G. Perboli, and R. Tadei. Extreme point-based heuristics for three-dimensional bin packing. *INFORMS Journal on Computing*, 20(3):368–384, 2008. doi: 10.1287/ijoc.1070.0250.

66. C. Johnson. What is research in computing science? *Computer Science Dept., Glasgow University. Electronic resource: https://www.dcs.gla.ac.uk/~johnson/teaching/research_skills/research.html*. URL https://www.dcs.gla.ac.uk/~johnson/teaching/research_skills/research.html. Accessed: 30 Oct 2023.

67. B. Oates, M. Griffiths, and R. McLean. *Researching Information Systems and Computing*. SAGE Publications, 2022. ISBN 9781529784930.

68. P. Schwerin and G. Wäscher. The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp. *International Transactions in Operational Research*, 4 (5-6):377–389, November 1997. ISSN 0969-6016.

69. J. Schoenfield. Fast, exact solution of open bin packing problems without linear programming. Technical report, US Army Space and Missile Defense Command, Huntsville, Alabama, USA, 2002.

70. G. Wäscher and T. Gau. Heuristics for the integer one-dimensional cutting stock problem: A computational study. *Operations-Research-Spektrum*, 18(3):131–144, September 1996. ISSN 1436-6304.

71. E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Proceedings of the IEEE 1992 International Conference on Robotics and Automation*, pages 1186–1192, 1992.

72. R. Vaessens, E. Aarts, and J. Lenstra. A local search template. *Computers & Operations Research*, 25(11):969–979, 1998. ISSN 0305-0548. doi: 10.1016/S0305-0548(97)00093-2.

73. M. Mısır, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe. Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010. doi: 10.1109/CEC.2010.5586348.

74. M. Gerber. Automated design of the deep neural network pipeline. Master's thesis, University of Pretoria, Department of Computer Science, 2021.

75. J. Levine and F. Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55(7):705–716, July 2004. doi: 10.1057/palgrave.jors.2601771.

76. V. Paschos. *Concepts of Combinatorial Optimization*. ISTE. Wiley, 2014. ISBN 9781119015079.

77. M. Hyde, G. Ochoa, T. Curtois, and J. Vázquez-Rodríguez. A hyflex module for the one dimensional bin-packing problem. Technical report, University of Nottingham, School of Computer Science, 2009.