

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY  
CAMPUS MONTERREY

SCHOOL OF ENGINEERING AND INFORMATION TECHNOLOGIES  
GRADUATE PROGRAMS



DOCTOR OF PHILOSOPHY

in

INFORMATION TECHNOLOGIES AND COMMUNICATIONS  
MAJOR IN INTELLIGENT SYSTEMS

**An Evolutionary Framework for Producing Hyper-heuristics for Solving  
the 2D Irregular Bin Packing Problem**

By

**Eunice López Camacho**

MAY 2012

# An Evolutionary Framework for Producing Hyper-heuristics for Solving the 2D Irregular Bin Packing Problem

A dissertation presented by

**Eunice López Camacho**

Submitted to the  
Graduate Programs in Electronics and Information Technologies  
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy  
in  
Information Technologies and Communications  
Major in Intelligent Systems



Thesis Committee:

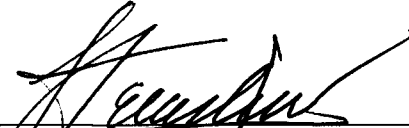
Dr. Hugo Terashima Marín - ITESM, Campus Monterrey  
Dr. Gabriela Ochoa - University of Nottingham  
Dr. Peter Ross - Edinburgh Napier University  
Dr. Eduardo Uresti Charre - ITESM, Campus Monterrey  
Dr. Manuel Valenzuela Rendón - ITESM, Campus Monterrey


Instituto Tecnológico y de Estudios Superiores de Monterrey  
Campus Monterrey  
May 2012


Instituto Tecnológico y de Estudios Superiores de Monterrey  
Campus Monterrey


School of Engineering and Information Technologies  
Graduate Program

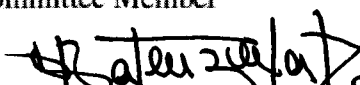
The committee members, hereby, certify that have read the dissertation presented by Eunice López Camacho and that it is fully adequate in scope and quality as a partial requirement for the degree of **Doctor of Philosophy in Information Technologies and Communications**, with a major in **Intelligent Systems**.


  
\_\_\_\_\_  
Dr. Hugo Terashima Marin  
ITESM, Campus Monterrey  
Principal Advisor

  
\_\_\_\_\_  
Dr. Gabriela Ochoa  
University of Nottingham  
Committee Member

  
\_\_\_\_\_  
Dr. Peter Ross  
Edinburgh Napier University  
Committee Member

  
\_\_\_\_\_  
Dr. Eduardo Uresti Charre  
ITESM, Campus Monterrey  
Committee Member

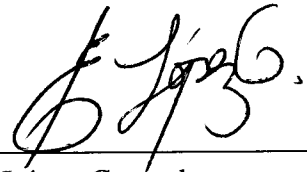
  
\_\_\_\_\_  
Dr. Manuel Valenzuela Rendon  
ITESM, Campus Monterrey  
Committee Member

  
\_\_\_\_\_  
Dr. José Luis Gordillo Moscoso  
Director of the Doctoral Program in Information Technologies and Communications  
School of Engineering and Information Technologies



# Copyright Declaration

I, hereby, declare that I wrote this dissertation entirely by myself and, that, it exclusively describes my own research.



---

Eunice López Camacho  
Monterrey, N.L., México  
May 2012

©2012 by Eunice López Camacho  
All Rights Reserved



# Dedication

To my husband Arturo

To my children Arturo, Pablo and Rebeca

To my parents Salomón and Felícitas

Thanks for all your unconditional confidence, support, patience, and encouragement.

You were my main motivation for pushing through this work.





# Acknowledgements

I would like to express my deepest gratitude to all those who have been side by side with me. Many people have played an important role in the successful completion of the research presented here. The number of people to whom I want to express my gratitude is incredible large, since I have been supported in many different ways.

I would like to express my sincere thanks to my advisors and committee members who have provided too much help and excellent guidance with the preparation of this work. Very special thanks to Dr. Hugo Terashima, Dr. Gabriela Ochoa and Dr. Peter Ross. They devoted much time guiding me. I would also like to thank to my fellow researchers: Dr. Juan Carlos Gómez and Dr. José Carlos Ortiz.

My husband, Arturo Rodríguez, have supported me any time that I have needed, and more. Without his love, help and advice, I would not have completed this dissertation. My parents also have provided me a great support, especially during the time I did my research stay. The love of my children was a great inspiration all over the time.

I thank Tecnológico de Monterrey to trust me and for gave me the opportunity to grow as a professional. Special thanks to the Research Chair on Evolutionary Computing CAT-144, the CONACYT project grant 99695, the Mathematics Department and the Planning and Organizational Development Department for all the support and confidence.



# **An Evolutionary Framework for Producing Hyper-heuristics for Solving the 2D Irregular Bin Packing Problem**

by

Eunice López Camacho

## **Abstract**

This document presents a doctoral dissertation which is a requirement for the Ph.D. degree in Information Technologies and Communications from Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM), Campus Monterrey, major in Intelligent Systems in the field of Hyper-heuristic Search for the Bin Packing Problem. This dissertation works with an evolutionary framework that produces hyper-heuristics for solving several types of bin packing problems introducing relevant improvements to the solution model.

The Bin Packing Problem is a particular case of the Cutting and Packing Problem, where a set of pieces are placed into identical objects and the objective is to minimize the number of objects needed. Given the NP-hard nature of this optimization problem, many heuristic approaches have been proposed. In this work a solution model is proposed, based on a genetic algorithm, in which a hyper-heuristic is built as a rule or a high-level heuristic that combines several low-level heuristics when building a solution from scratch. Therefore, the hyper-heuristic takes advantage of the main strengths of the low-level heuristics to solve particular kinds of problem instances. A hyper-heuristic is a list of several representative states of the problem, each one labelled by a low-level heuristic. A problem instance to be solved by a hyper-heuristic is first summarized by a numerical vector that carries some of its main features. This vector is then compared with the hyper-heuristic representative states and the correspondent heuristic is chosen to be applied. After one or several pieces are placed, the problem state is updated. This process continues until the problem instance is completely solved.

The main inputs of the evolutionary framework in order to produce a hyper-heuristic are: (1) a vector-based way of representing the problem state; (2) a set of low-level heuristics; and (3) a set of training problem instances. The research presented in this document improves each of these elements.

First, a data mining based methodology was developed to select the best set of features that can represent the state of the problem instances. This six-step methodology includes the application of the  $k$ -means clustering technique and a Multinomial Logistic Regression model to find a subset of features that best predict heuristic performance. This methodology does not require intensive knowledge of the problem domain. Promising results were found when comparing hyper-heuristics produced employing an intuitive representation and a representation built with this methodology. Besides, there are some other solution approaches developed for other combinatorial optimization problems that also require to represent instances with a limited number of features. Therefore, the proposed methodology can be exported for other

search space approaches.

Second, the Djang and Finch selection heuristic was properly adapted from the one-dimensional to the two-dimensional Bin Packing Problem. This adaptation includes a time-saving routine based on avoiding repetitive computations. With the pieces in decreasing order, this heuristic starts filling an object until it is at least one-third full. Then it tries combinations of one, two or three pieces that completely fills the object. If this is not possible, a small waste is allowed and it is increased as necessary until there are not remaining pieces that fit. Then, a new object is opened. Several experiments were conducted with the initial fraction of the object that is full before trying combination of pieces. We found that filling the object up to one-fourth, one-third and one-half produces effective heuristics that behaves differently in different kinds of instances.

Third, the level of generality handled by the evolutionary framework was increased in terms of the kind of instances solved. The solution model can be trained with one- and two-dimension regular and irregular instances. Irregular instances include convex and concave polygons. Once the hyper-heuristic is evolved, it is able to solved any instance from any of these types with good results and without further parameter tuning. The framework was tested with a large dataset of 1417 instances. One-dimensional instances were drawn from the literature. An algorithm was designed for randomly producing two-dimensional instances with concave pieces. Therefore, geometric functions had to be implemented for dealing with concavities. Twenty hyper-heuristics were generated and tested. Broadly speaking, hyper-heuristics were able to learn a combination of single heuristics that produce, at least, the same result than the best single heuristic for each testing case.

Finally, an analysis was performed to find which feature values of the Bin Packing Problem are more likely to lead to a good performance of heuristics and hyper-heuristics. With the Principal Component Analysis technique, a two-dimensional map was built in which the 1417 instances were plotted. The more similar two instances are according to nine selected features, the closer the two instances are plotted in the map. It is possible to find combination of feature values that characterize each section of the map. By over imposing the performance of heuristics and hyper-heuristics over the map, we can draw conclusions about the main relations among features and performance. Understanding the Bin Packing Problem structure will help in the design of new solution approaches.

# List of Figures

1.1	General scheme of the 1D Bin Packing Problem. . . . .	4
1.2	General scheme of the 2D Irregular Bin Packing Problem. . . . .	5
1.3	Simplified representation of a problem state. . . . .	7
1.4	General structure of the solution model studied. Fonts in <i>italics</i> explain the main improvements to the solution model. . . . .	9
2.1	Basic and intermediate problem types for the C&P problem in Wäscher’s Typology [154]. . . . .	17
2.2	Interpretation of the $D$ -function. . . . .	22
2.3	Generation of the NFP between a square (fixed polygon) and a triangle (orbiting polygon). . . . .	23
2.4	(a) The orientation of $A$ and $B$ ; (b) the slopes diagram; and (c) the NFP. . . . .	23
2.5	Bottom-left heuristic for irregular pieces. . . . .	28
2.6	Positions to be considered in the Constructive Approach. . . . .	28
2.7	Positions to be considered in the Constructive Approach. An example with two pieces already placed. . . . .	29
2.8	Constructive Approach Heuristic. . . . .	29
2.9	Rectangle with minimum area, located at the bottom-left corner and containing all pieces so far. . . . .	30
2.10	Candidate rectangles when locating a piece. . . . .	30
2.11	Two concave pieces that can not fit with a sliding operation. . . . .	31
3.1	Solution Model for Hyper-heuristic generation through GA. . . . .	46
3.2	A chromosome is a set of blocks. . . . .	48
3.3	A chromosome is a set of blocks. Each block represents a point in the hypercube (space of states) labelled with a single heuristic. The solution process of a problem using a hyper-heuristic consists of finding the closest single heuristic at every solution stage. . . . .	48
3.4	General process of the genetic algorithm. . . . .	51
3.5	An example of the two-point crossover operator. . . . .	52
3.6	Representation of problem instances in a text file. . . . .	54
3.7	Origin of the coordinates of the pieces of a problem instance. . . . .	54
4.1	Degree of concavity. . . . .	62
4.2	A piece and its convex hull. . . . .	62
4.3	Procedure for selecting a point inside a shape. . . . .	64

4.4	One convex and one non-convex polygon created by the developed algorithm.	64
4.5	(a) A convex problem instance. (b) 5 pieces were randomly selected to build 5 non-convex polygons. . . . .	65
4.6	Ray from point $P$ to the right actually touches 4 times the shape boundaries. The ray crosses the shape at vertex $B$ . In contrast, the ray touches vertex $A$ only tangentially and does not cross the shape at this point. Therefore, the count for crosses is 3. Since 3 is an odd number, we conclude that $P$ is inside the shape. . . . .	70
4.7	Piece $AEFG$ is inside piece $ABCDEFG$ . In this case, checking if all vertices and edges midpoints of $AEFG$ are inside $ABCDEFG$ will return <b>false</b> . Only when a point very close to vertex $E$ is found inside $ABCDEFG$ , the algorithm returns <b>true</b> to the question about if one of the pieces is inside the other. In this case, reviewing intersection of these two pieces with Algorithm 1 will return <b>false</b> because none of the sides crosses another (although they coincide). . . . .	72
4.8	Piece $ABCDEFGH$ contains all area to the (a) left and (b) below a given piece. . . . .	72
6.1	Matrix of normalized performance. . . . .	83
6.2	General idea for identifying those features related with heuristics performance. First, instances are clustered according to heuristic performance and then features are related with cluster membership. . . . .	86
6.3	Lineal mapping to the interval $[0, 1]$ for feature measures. . . . .	89
6.4	Normalized performance of heuristics BF+BLI ( $h_1$ ) and BFD+CA ( $h_2$ ) related with instances mean number of sides. . . . .	91
6.5	Normalized performance of heuristics BF+BLI ( $h_1$ ) and BFD+CA ( $h_2$ ) related with the instance variance of the number of sides. . . . .	91
6.6	Average fitness of the whole population for all experiments conducted under each representation scheme. . . . .	94
7.1	DJD <sub>1/3</sub> in combination with CAD heuristic solves an instance of type $C$ . This solution needs one more object than the optimal solution in which there would be zero waste. In this solution, fitness is 0.776 measured with equation 3.2 . . .	104
7.2	Comparison of means for the 11 heuristics considered, using the Bonferroni adjustment. DJD <sub>1/4</sub> and DJD <sub>1/3</sub> are the better heuristics and there is not significant difference between them . . . . .	106
8.1	Number of pieces vs. computational time for the 397 one-dimensional instances. Six single heuristics. . . . .	127
8.2	Number of pieces vs. computational time for the 1020 two-dimensional instances. Six single heuristics. . . . .	128
8.3	Number of pieces vs. computational time for the 80 one-dimensional instances of types $Trip60$ , $Trip120$ , $Trip249$ and $Trip501$ . Heuristic DJD <sub>1/3</sub> . The line is the graph of the cubic polynomial which fits best all the points. . . . .	129
8.4	Number of pieces vs. computational time for the average of the hyper-heuristics tested in each instance. 1D and 2D instances. . . . .	129

9.1	An example of principal component analysis for a dataset of 2 variables. (a) PCA identifies the two orthogonal directions (PC1 and PC2) along which the data have the largest spread. (b) Observations plotted in one dimension using their projections onto PC1. . . . .	133
9.2	(a) The 1417 instances plotted along PC1 and PC2. (b.1) The 397 instances of the 1D BPP. (b.2) The 540 instances of the 2D BPP (convex). The 30 rectangular instances are all plotted almost in the same place inside the circle. (b.3) The 480 instances of the 2D irregular BPP (non-convex). . . . .	135
9.3	PCA 2D-loading plot of the two first principal components. . . . .	136
9.4	Distribution of feature values across the PCA map (black shows maximum value for each feature). Horizontal axis is for PC1 score, while vertical axis is for PC2 score. . . . .	138
9.5	Performance of the 6 heuristics across all instances in the PCA map (black shows maximum value). Horizontal axis is for PC1 score, while vertical axis is for PC2 score. . . . .	139
9.6	Normalized performance of the 6 heuristics across all instances in the PCA map (black shows maximum value). Horizontal axis is for PC1 score, while vertical axis is for PC2 score. . . . .	140
9.7	Instances in each of the 8 clusters built based on the normalized performance. Horizontal axis is for PC1 score, while vertical axis is for PC2 score. . . . .	141
9.8	Best heuristic. Color grey is for instances better solved by 1-piece heuristics (FFD, Filler or BFD). Points in color black are instances better solved by the DJD heuristics. . . . .	142
9.9	From all instances, only 28 were solved with fewer objects by the best hyper-heuristic than any of the 6 heuristics (plotted with the letter <b>b</b> ). Whilst in 9 cases, none hyper-heuristic could reach the best single heuristic result (plotted with the letter <b>w</b> ). In all the remaining cases, the best hyper-heuristic got the same number of objects than the best heuristic for each case (gray dots). . . . .	142





# List of Tables

4.1	Description of problem instances. . . . .	59
4.2	Characteristics of the 1D problem instances . . . . .	60
4.3	Characteristics of the convex 2D problem instances. . . . .	61
4.4	Characteristics of the non-convex 2D problem instances. . . . .	63
5.1	Representation of actions. . . . .	75
5.2	Number of extra objects for the testing set and compared against results of the best single heuristics. Experiment I, for first and second independent runs. Figures in last 10 columns average the performance of four heuristics. These heuristics are indicated by the numbers under the selection heuristic name (see Table 5.1). . . . .	77
5.3	Number of extra objects for the testing set and compared against results of the best single heuristics. Experiment II. Figures in last 10 columns average the performance of four heuristics. These heuristics are indicated by the numbers under the selection heuristic name (see Table 5.1). . . . .	77
5.4	Number of extra objects for the testing set and compared against results of the best single heuristics. Experiment III. Figures in last 10 columns average the performance of four heuristics. These heuristics are indicated by the numbers under the selection heuristic name (see Table 5.1). . . . .	78
5.5	Number of extra objects for the testing set and compared against the best results of the best single heuristics. Experiment IV. Figures in last 10 columns average the performance of four heuristics. These heuristics are indicated by the numbers under the selection heuristic name (see Table 5.1). . . . .	78
5.6	Number of extra objects used by the hyper-heuristics in the testing set with respect to the average obtained by the 40 single heuristics. . . . .	79
6.1	Representation <b>1</b> of the instance state. . . . .	82
6.2	Representation <b>2</b> of the instance state. . . . .	90
6.3	Number of extra objects delivered by hyper-heuristics using representation scheme 2 compared with those using representation scheme 1. Experiments I through IV. . . . .	93
6.4	Average number of extra objects obtained by hyper-heuristics using representation scheme 2 compared with those using representation scheme 1. . . . .	94
7.1	Average fitness for all the combinations of selection and placement heuristics over the 540 instances. . . . .	104

7.2	Average fitness obtained by all the selection heuristics when combined with the CAD placement heuristic for each instance type. The best selection heuristic for each instance type is in bold font. . . . .	105
7.3	Average computational time (in seconds) for all the combinations of selection and placement heuristics over the 540 instances. . . . .	105
8.1	Number of instances associated to each cluster for all the instance types considered. The clusters are obtained according to the fitness of the six single heuristics selected. . . . .	113
8.2	Cluster membership for the 1D and 2D instances. According to fitness of the six single heuristics considered. . . . .	114
8.3	Representation of the instance state. . . . .	115
8.4	Hyper-heuristic generated in the first run of Experiment 1. . . . .	116
8.5	Number of extra objects obtained by hyper-heuristics and single heuristics compared against results of the best single heuristic for each instance (percentage of cases). Zero values are displayed as blank cells. . . . .	117
8.6	Average number of extra objects delivered by the best hyper-heuristic, compared against results of the best single heuristic for each instance. . . . .	118
8.7	Percentage of selection of each single heuristic when solving the testing set with hyper-heuristics of Experiment 1. . . . .	119
8.8	Solving non-convex instances compared when solving their convex version. Percentage of cases when non-convex instances require fewer, equal and more objects than convex instances. Instances are solved by the best single heuristic and by the best hyper-heuristic from experiments 1 through 4. . . . .	120
8.9	Percentage of single heuristic changes when solving all testing sets. . . . .	120
8.10	Average of heuristic changes according to the number of extra objects against results of best single heuristic. . . . .	121
8.11	Average length of single heuristic sequences when applying our hyper-heuristic solution model. . . . .	122
8.12	Percentage of sequences of single heuristic pairs when solving 1D instances in all testing sets. . . . .	122
8.13	Percentage of sequences of single heuristic pairs when solving 2D convex instances in all testing sets. . . . .	123
8.14	Percentage of sequences of single heuristic pairs when solving 2D non-convex instances in all testing sets. . . . .	123
8.15	Average computational time (in seconds) per category of instances. . . . .	124
8.16	Average computational time (in seconds) per type of instances. . . . .	125
9.1	Loadings for the two main principal components of the data. Features 1 through 9 are those referred in Table 8.3. Figures with largest absolute values are in bold font. . . . .	135

# List of Algorithms

1	Decide if two pieces intersects each other. . . . .	66
2	Decide if a point is inside a shape. . . . .	67
3	Decide if a shape is completely inside another shape. . . . .	68
4	Measures the distance in which two segments coincide. . . . .	69
5	Builds a piece containing all the area at the left of a given piece. . . . .	69
6	Computes the horizontal distance from a point to a given segment. Distance is zero if the point is along the segment. Distance is positive if the point is in the right of the segment. Otherwise it is negative. . . . .	71
7	Computes the distance that a given piece can be moved to the left without overlapping other pieces and without exceeding the object limits. . . . .	71
8	The original DJD heuristic. . . . .	98
9	The proposed DJD algorithm. Trying pieces one by one. . . . .	100
10	The proposed DJD algorithm. Trying groups of 2 pieces. . . . .	100
11	The proposed DJD algorithm. Trying groups of 3 pieces. . . . .	101



# Contents

<b>Abstract</b>	<b>ix</b>
<b>List of Figures</b>	<b>xiii</b>
<b>List of Tables</b>	<b>xvi</b>
<b>List of Algorithms</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Statement and Context . . . . .	3
1.2 Motivation . . . . .	5
1.3 Hypothesis and Research Questions . . . . .	7
1.4 Solution Overview . . . . .	8
1.5 Main Contributions . . . . .	10
1.6 Scope and Limitations . . . . .	11
1.7 Thesis Organization . . . . .	12
<b>2 Related Background</b>	<b>13</b>
2.1 Cutting and Packing (C&P) Problems . . . . .	13
2.1.1 Dyckhoff’s Typology . . . . .	14
2.1.2 Wäscher’s Typology . . . . .	15
2.2 Industrial Applications for the C&P Problem . . . . .	16
2.2.1 Industrial Constraints . . . . .	18
2.3 Packing and NP-Hardness . . . . .	19
2.4 The Geometry for Placing Irregular Polygons . . . . .	21
2.4.1 Pixel/raster Method . . . . .	21
2.4.2 Direct Trigonometry . . . . .	21
2.4.3 The No-Fit Polygon (NFP) . . . . .	22
2.4.4 The Phi Function . . . . .	24
2.5 Exact Solutions for the 2D C&P Problem . . . . .	24
2.5.1 Linear Programming . . . . .	24
2.5.2 Tree-Search Procedures . . . . .	24
2.6 Heuristic Search for the 2D C&P Problem . . . . .	25
2.6.1 Selection Heuristics . . . . .	26
2.6.2 Placement Heuristics . . . . .	27
2.6.3 Local Search Heuristics . . . . .	31

2.7	Meta-heuristic Search for the 2D C&P Problem . . . . .	31
2.7.1	Evolutionary Computation (EC) . . . . .	32
2.7.2	Simulated Annealing . . . . .	34
2.7.3	Tabu Search . . . . .	34
2.7.4	Ant Colony Optimization (ACO) . . . . .	34
2.7.5	Greedy Randomized Adaptive Search Procedure (GRASP) . . . . .	35
2.7.6	Agent-Based Approach . . . . .	35
2.8	Hyper-heuristic Search . . . . .	36
2.8.1	Heuristic Selection: Iterative Hyper-heuristics . . . . .	37
2.8.2	Heuristic Selection: Constructive Hyper-heuristics . . . . .	38
2.8.3	Heuristic Generation . . . . .	38
2.9	Hyper-heuristic Search for the 2D C&P Problem . . . . .	39
2.9.1	Hyper-heuristics with Evolutionary Computation . . . . .	39
2.9.2	Learning Classifier Systems . . . . .	39
2.9.3	Genetic Programming . . . . .	39
2.9.4	Genetic Algorithms . . . . .	40
2.9.5	Hyper-heuristics with Ant Colony Optimization . . . . .	40
2.9.6	Hyper-heuristics with Hill Climbing . . . . .	40
2.10	Hyper-heuristic Search for other Optimization Problems . . . . .	41
2.10.1	Timetabling Problems . . . . .	41
2.10.2	Constraint Satisfaction Problems (CSP) . . . . .	42
2.11	Meta-learning . . . . .	42
2.12	Summary . . . . .	43
<b>3</b>	<b>Hyper-heuristic Solution Model</b>	<b>45</b>
3.1	General Process for Generating Hyper-heuristics . . . . .	46
3.2	Representation of Problem Instances . . . . .	47
3.3	Representation of Chromosomes in the GA . . . . .	47
3.4	Fitness Function . . . . .	49
3.5	The GA Cycle . . . . .	50
3.5.1	Genetic Operators . . . . .	50
3.6	Rotation Scheme . . . . .	53
3.7	Codification of Problem Instances . . . . .	53
3.8	Summary . . . . .	53
<b>4</b>	<b>Research Methodology and Experimental Setup</b>	<b>55</b>
4.1	Methodology . . . . .	55
4.2	Problem Instances Testbed . . . . .	58
4.2.1	1D Instances . . . . .	58
4.2.2	Convex 2D Instances . . . . .	59
4.2.3	Non-convex 2D Instances . . . . .	60
4.2.4	Algorithm for Producing Random Instances that Include Non-convex Pieces . . . . .	62
4.3	Algorithms Developed for Geometric Computation . . . . .	64

4.4	Summary . . . . .	70
<b>5</b>	<b>Hyper-heuristics for 2D Irregular BPP (Convex)</b>	<b>73</b>
5.1	Experimental Setup . . . . .	73
5.1.1	Set of heuristics . . . . .	73
5.1.2	Chromosomes . . . . .	73
5.1.3	Representation of problem instance state . . . . .	74
5.1.4	Testbed Instances . . . . .	74
5.2	Experiments . . . . .	74
5.3	Results and Discussion . . . . .	76
5.4	Summary . . . . .	79
<b>6</b>	<b>Defining a Problem-State Representation Scheme</b>	<b>81</b>
6.1	Methodology for Developing a Representation Scheme . . . . .	83
6.2	Developing a New Representation Scheme . . . . .	86
6.3	Experiments . . . . .	91
6.4	Results and Discussion . . . . .	93
6.5	Summary . . . . .	94
<b>7</b>	<b>An Effective Heuristic for the 2D Irregular BPP</b>	<b>97</b>
7.1	The DJD Heuristic . . . . .	98
7.2	The Proposed DJD Heuristic for the 2D Irregular BPP . . . . .	99
7.3	Experiments . . . . .	102
7.3.1	The Other Selection Heuristics and the Placement Heuristics . . . . .	102
7.4	Results and Discussion . . . . .	103
7.5	Summary . . . . .	106
<b>8</b>	<b>Hyper-heuristics for 1D and 2D Bin Packing Problems</b>	<b>109</b>
8.1	Implementation . . . . .	110
8.1.1	Developing a Problem-state Representation for the Testbed Instances . . . . .	111
8.2	Experimental Design . . . . .	114
8.3	Results and Discussion . . . . .	115
8.3.1	Comparison Against the Best Single Heuristic . . . . .	116
8.3.2	Analyzing Results per Instance Category . . . . .	118
8.3.3	Comparing Results for Convex and Non-convex Instances . . . . .	119
8.3.4	Alternation of Single Heuristics . . . . .	120
8.3.5	Time Complexity . . . . .	122
8.4	Summary . . . . .	126
<b>9</b>	<b>A Deeper Understanding of the BPP Structure</b>	<b>131</b>
9.1	Principal Component Analysis (PCA) . . . . .	132
9.2	Experimental Setup . . . . .	133
9.2.1	Meta-data for the Bin Packing Problem . . . . .	134
9.3	Results and Discussion . . . . .	134
9.3.1	Distribution of Features across the PCA Map . . . . .	137

9.3.2	Distribution of Heuristic Performance Across the PCA Map . . . . .	137
9.3.3	Clustering . . . . .	137
9.3.4	The Best Heuristic . . . . .	138
9.3.5	Hyper-heuristic Performance . . . . .	140
9.4	Summary . . . . .	143
<b>10</b>	<b>Final Conclusions and Future Work</b>	<b>145</b>
10.1	Dissertation Summary and General Discussion . . . . .	145
10.1.1	Hyper-heuristics for Solving 2D Irregular Bin Packing Problems (convex) . . . . .	145
10.1.2	Defining a Problem-State Representation Scheme . . . . .	145
10.1.3	The DJD Heuristic as an Effective Heuristic for the 2D Irregular BPP	146
10.1.4	Hyper-heuristics for Solving 1D and 2D Irregular Bin Packing Problems (concave) . . . . .	146
10.1.5	A Knowledge Discovery Approach for Understanding how Features Impact in Heuristic and Hyper-heuristic performance . . . . .	147
10.1.6	General Discussion . . . . .	147
10.2	Main Contributions . . . . .	148
10.2.1	Hyper-heuristics for Solving 2D Irregular Bin Packing Problems (convex) . . . . .	148
10.2.2	Defining a Problem-State Representation Scheme . . . . .	148
10.2.3	The DJD Heuristic as an Effective Heuristic for the 2D Irregular BPP	148
10.2.4	Hyper-heuristics for Solving 1D and 2D Irregular Bin Packing Problems (concave) . . . . .	148
10.2.5	A Knowledge Discovery Approach for Understanding how Features Impact in Heuristic and Hyper-heuristic performance . . . . .	149
10.3	Future Direction . . . . .	149
10.4	Closing Remarks . . . . .	150
<b>A</b>	<b>Contributed Scientific Publications</b>	<b>151</b>
	<b>Bibliography</b>	<b>165</b>



# Chapter 1

## Introduction

The problem of finding an arrangement of pieces to cut or pack inside larger objects is known as the **Cutting and Packing (C&P) Problem**. This problem has huge practical relevance since it has a large number of practical applications. The **two-dimensional Bin Packing Problem (2D BPP)** is a particular case of the C&P problem, which consists of finding an arrangement of pieces inside the identical objects such that the number of objects required to contain all pieces is minimum. The case of rectangular pieces is the most studied one; although, there are many instances where irregular patterns arise, for example the metal, shoe and garment manufacturing. There is not a polynomial-time algorithm to solve optimally the 2D BPP, that is why heuristics are developed to find a good, frequently suboptimal, solutions.

A heuristic (from Greek *heuriskein*, *to find*) is a practical rule or procedure that tries to exploit problem-specific knowledge and usually comes from a logical idea with common sense, although its goodness is not formally proved [9]. For many practical applications, an optimal solution is not absolutely necessary. Sometimes, instead of a solution very close to the optimum, it is preferred a *good enough - soon enough - cheap enough* solution [22]. An approximation algorithm, as well as a heuristic, does not guarantee to find the best solution, but a good enough solution instead in a time polynomially bounded. The difference between a heuristic and an approximation algorithm is that the latter give us a guarantee as to how bad solutions we can get. Normally specified as  $c$  times the optimal value [108].

Moreover, some heuristics are very problem-specific and their performance is rather poor in a different instance domain. There is a spectrum which ranges from cheap but fragile heuristics at one extreme and knowledge-intensive methods that can perform very well but are hard to implement and maintain, at the other extreme. Nevertheless, even complex heuristics may face problem instances where they perform badly. In general, some methods work well for particular instances, but not for all of them. Hyper-heuristics is an emerging search technology that is motivated, to a large extent, by the goal of raising the level of generality at which optimization systems can operate [22]. Hyper-heuristics might be thought as heuristics or algorithms to choose or generate heuristics. The idea behind hyper-heuristics is to discover some combination of straightforward heuristics to solve a wide range of problems, taking advantage on the strength each heuristic may have in a particular kind of problems. Raising the level of generality can be interpreted as building an optimization algorithm that tackles: (1) a wider range of instances from the same problem still producing good enough solutions, and/or (2) a wider spectrum of combinatorial problems with little implementation effort.

Terashima et al. [143] described a genetic-algorithm-based method that produces hyper-heuristics to solve two-dimensional regular (rectangular) and irregular (convex polygonal) Bin Packing problems. This solution model is the building block for this dissertation and it is described in Chapter 3, while the implementation for the 2D BPP with convex polygons is presented in Chapter 5. The genetic algorithm included in the model uses a variable-length representation, producing hyper-heuristics after going through a learning process which includes training and testing phases. Such hyper-heuristics constructed by the evolutionary procedure are sets of condition-action rules, where the condition is a numerical vector that represent the state of the problem taking into account problem relevant features. The action of every rule is one of the available low-level heuristics, and it is the heuristic to be applied. After applying the corresponding heuristic, the problem of placing the remaining pieces is again characterized by a vector and, maybe, another low-level heuristic is applied to reduce the number of pieces remaining to be placed. This process continues until every piece is placed solving the problem completely.

This doctoral dissertation looks deeper into this solution model, investigating four important aspects, which are developed in Chapters 6 to 9.

- Improvement of the numerical problem instance representation. Each instance to be solved by the hyper-heuristic is characterized by a numerical vector that summarizes some of its relevant features. Each numerical term of the vector quantifies an aspect of the instance at hand, for example, number of pieces, percentage of small pieces or average number of pieces sides, etc. According to this numerical vector, the hyper-heuristic decides which low-level heuristic to apply every time. The 2D irregular BPP may have many features and not all of them are relevant in the selection of a low-level heuristic, so not all of them should be included in the problem state representation. An adequate problem state representation scheme is critical for the good performance of the hyper-heuristic. The procedure for designing the problem representation deserves attention. Moreover, this investigation creates a general methodology for building a problem representation without much knowledge about the problem properties (see Chapter 6).
- Generalization of the solution model through the kind of problem instances that the solver can handle. The model that constructs hyper-heuristics for the 2D BPP accepts instances that are either rectangles or irregular convex polygons. After obtaining good results with this kind of instances [143], the following natural step is to extend the methodology to instances that include concave polygons (at least one internal angle greater than 180 degrees). This involves additional geometric complexity increasing the computational burden. To make the solution framework even more general, the 1D BPP is included as well. So, hyper-heuristics generated are able to choose the best single heuristics to any problem instance from 1D to 2D irregular (non-convex) without human intervention. Chapter 8 develops this framework in detail.
- Analysis and improvement of the repository of low-level heuristics. The quality and diversity of the low-level heuristics combined by a hyper-heuristic is crucial. On one hand, we choose one of the single heuristics of the model and improved its performance making an adaptation for the 2D case in such a way that is fast in execution and delivers high quality solutions. This heuristic, called Djang and Finch (DJD), is designed for

selecting which pieces are the next to be placed (Chapter 7). On the other hand, an analysis of the repository of low-level heuristics is performed expecting to obtain a better understanding of how the interaction of heuristics works during the resolution of a problem instance. Sequence, frequency and alternation of heuristics are important issues analyzed in Sections 8.3.2 and 8.3.4.

- Analysis of the structure of the Bin Packing Problem and its relation with heuristics and hyper-heuristics performance. The BPP, specially the 2D irregular case, has a complex structure, since it can be characterized by many features. Insights about which combination of features have impact in heuristic and hyper-heuristic performance are developed in Chapter 9, where the mathematical technique called Principal Component Analysis is employed to visually assess multivariate data in two dimensions.

In the next section the problem is stated in a more specific and restricted way, describing the specific problem under study along with the objectives that are sought for in this project. Section 1.2 presents a more thorough explanation of what motivates this research. After, the hypothesis that supports all this research is presented plus the research questions to which an answer is being looked for. Following, an overview of the proposed solution is presented along with the main contributions that this research project has provided. Scope and limitations of this dissertation are presented next. Finally, a description of how the dissertation has been organized is laid out.

## 1.1 Problem Statement and Context

The main problem to work with during the investigation is stated in this section. The Cutting and Packing Problem is among the earliest problems in the literature of operational research. Due to the extensive work done in this NP-problem, in 2006 Wäscher et al. [154] suggested a complete typology which is an extension of Dychoff's [48]. In the 1D BPP, there is an unlimited supply of bins, each with capacity  $c > 0$ . A set of  $n$  items (each one of size  $s_i < c$ ) is to be packed into the bins. The task is to minimize the total number of bins used. In the 2D BPP, there is a set  $L = (a_1, a_2, \dots, a_n)$  of pieces to cut or pack and an infinite set of identical rectangular larger elements (called *objects*) with dimensions  $x_0$  and  $y_0$ . Then, the 2D BPP consists of finding an arrangement of *pieces* inside the *objects* such that the number of objects required to contain all pieces is minimum. A feasible solution is an arrangement of pieces without overlaps and with no piece outside the object limits. A *problem instance* or *instance*  $I = (L, x_0, y_0)$  consists of a list of elements  $L$  and object dimensions  $x_0$  and  $y_0$ . An *instance state* is every intermediate phase in the solution process until all pieces are placed and a solution for the instance is found. The problem is called 2D irregular BPP when pieces are not rectangular [154].

A typical way of representing elements (and used in this research) is with a list of sets of coordinates  $L = (c(a_1), c(a_2), \dots, c(a_n))$ ; where  $c(a_i)$  is the set of coordinates in the Cartesian plane corresponding to piece  $a_i$ . A set of coordinates  $c(a_i)$  is a set of points represented by pairs  $(x, y)$ . For example:  $c(a_i) = \{(x_{i1}, y_{i1}), (x_{i2}, y_{i2}), \dots, (x_{ik_i}, y_{ik_i})\}$ , where  $k_i$  is the number of vertices of piece  $a_i$ . Each piece coordinates has an arbitrary reference (see Section 3.7).

The term 2D regular Bin Packing Problem (2D regular BPP) is mainly used when all pieces are rectangular (although circles and other regular shapes could fall under this name too [154]). Otherwise, the problem is called 2D irregular Bin Packing Problem (2D irregular BPP). The research presented in this dissertation is focused mainly on the 2D irregular BPP, but it also considers the 1D BPP in Chapters 8 and 9.

The case of rectangular pieces is the most widely studied. However, the irregular case is seen in a number of industries where parts with irregular shapes are cut from rectangular materials. For example, in the shipbuilding industry, plate parts with free-form shapes for use in the inner frameworks of ships are cut from rectangular steel plates, and in the apparel industry, parts of clothes and shoes are cut from fabric or leather [109]. Other direct applications include the optimization of layouts within the wood, metal, plastics, carbon fibre and glass industries.

For the 1D case, it is common to use the terms *items* and *bins*; whereas, for the 2D case, a variety of terms have been used. The small elements have been named *pieces*, *shapes* or *items*; and the large elements have been called *objects*, *stock* or *sheets*. In this investigation, we use the terms *items* and *bins* regarding the 1D case, and *pieces* and *objects* when referring to the 2D case.

This investigation considers pieces that are irregular polygons (sides and internal angles may not be equal) with unrestricted number of sides. We call 2D irregular BPP (convex) to the problem when all pieces are convex polygons (internal angles less than 180 degrees); when including some concave polygons as pieces, the problem is called 2D irregular BPP (non-convex) in this research. We assume the polygons are in *standard* form: no three consecutive vertices are collinear [145]. Pieces may rotate orthogonally in their attempt to fit inside an object. Cuts are infinitely thin. Figure 1.1 presents the 1D BPP, while Figure 1.2 shows an instance of the 2D irregular BPP (non-convex) problem, in which pieces are packed into two objects.

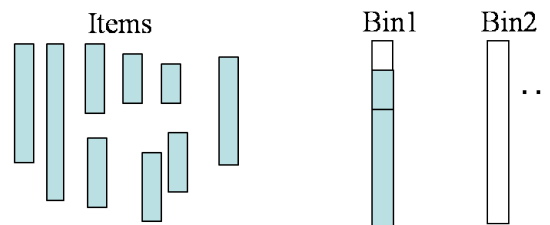


Figure 1.1: General scheme of the 1D Bin Packing Problem.

The Strip Packing Problem is a popular variant of the 2D Cutting and Packing problem which has only one large rectangular object with fixed width, its length is variable and has to be minimized after placing all the small pieces (some approaches and reviews in [44, 74, 23]). A variation of this typical strip packing problem consists of nesting irregular pieces in one large irregular object (examples in [93, 92, 155, 120]). The amount of research devoted to the strip packing problem has been larger when compared to the research about the 2D irregular BPP. We found that [109] studied a problem similar to our **2D irregular single bin size BPP** but using variable bin sizes, where the problem solution involves finding appropriate sizes of material objects (bins) among given standard sizes in order to reduce waste. We found

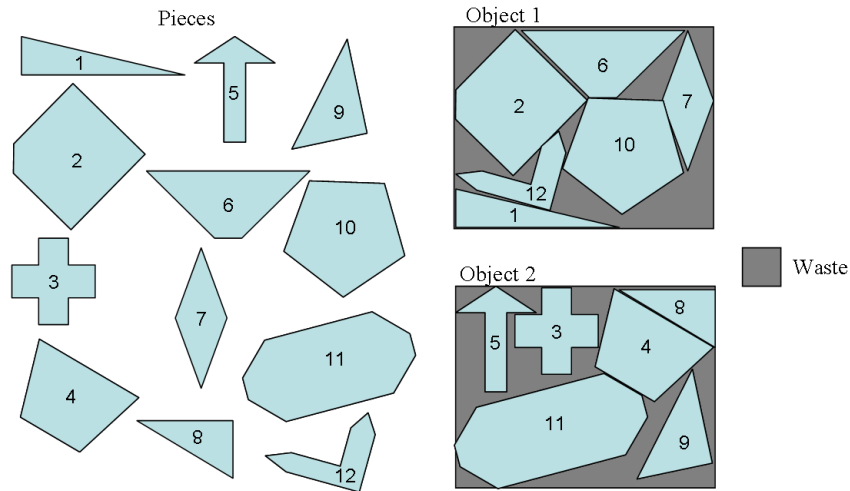


Figure 1.2: General scheme of the 2D Irregular Bin Packing Problem.

only one study about, specifically, the *2D irregular single bin size bin packing problem* where Ponce-Pérez et al. [117] proposed a genetic algorithm based approach. They tested it with one four-piece instance whose best result had some overlapping of pieces. As far as we know, there are no other studies for the *2D irregular single bin size bin packing problem*. Therefore, all 2D irregular problem instances in the literature are intended for the strip packing problem; and there are not 2D irregular BPP instances available. Also, since the strip packing problem is similar to the 2D irregular BPP, many heuristic implementations may be similar; although, results for both kind of problems are not comparable. Nevertheless the lack of research in the 2D irregular BPP, there exist many practical applications where irregular pieces are cut from identical rectangular objects [109].

The main objective of this work is to **increase the level of generality** of a solution model for the 2D irregular BPP based on an evolutionary hyper-heuristic approach. This main objective has the following building blocks: (1) developing a general scheme for finding a good representation of the problem state; (2) upgrading the model to solve instances with some concave polygons as well as 1D instances without requiring parameter tuning after a hyper-heuristic was developed; (3) improving the single heuristics repository and performing an analysis of the single heuristics inside the hyper-heuristic model; and (4) developing insights in the way that some features and combination of features impact heuristic and hyper-heuristic performance.

## 1.2 Motivation

The motivation to reach this problem comes mainly from the practical relevance the Bin Packing Problem has, especially when it is about two dimensional irregular pieces. Section 2.2 details the most important practical applications. Moreover, in most cases it is still difficult to outperform an experienced worker with a purely automatic algorithm, so that the irregular problem has become an attractive research topic [76]. Any kind of improvement towards the

resolution of this problem would benefit to the practical and industrial areas in which these type of problems appear.

On the other side, the motivation to utilize a hyper-heuristics approach comes from the fact that higher level of generality is becoming a trend in optimization. There is a current school of thought in meta-heuristic and search technology that states that one of the main goals of the discipline over the next few years is to raise the level of generality at which meta-heuristic and optimization systems can operate. For example, Burke et al. [20] conducted an empirical study that ran the same high-level strategies (hyper-heuristics) with three different domains: 1D bin packing, permutation flow shop and personnel scheduling. They used a software framework for the development of cross-domain search methodologies called Hyflex [21]. The term hyper-heuristic has been defined to broadly describe the process of using (meta-)heuristics to choose (meta-)heuristics to solve the problem in hand. One of the main motivations for studying hyper-heuristic approaches is that they should be cheaper to implement and easier to use than problem specific special purpose methods and the goal is to produce good quality solutions in this more general framework [22].

The *No free lunch* (NFL) theorem [156] establishes that all algorithms have the same average performance when considering all possible problems. In other words, for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class. An optimization problem, as defined by Wolpert and Mcready [156] is a function which maps a search space  $\mathcal{X}$  with an element of the space of possible *cost* values  $\mathcal{Y}$ . Both spaces, though huge, are finite according to the NFL statement. Their sizes are  $|\mathcal{X}|$  and  $|\mathcal{Y}|$ . The size of the space of all possible functions is  $|\mathcal{Y}|^{|\mathcal{X}|}$ . It is worth noting that many (or most) of this possible functions (problems) are trivial or unrealistic. For example, the instance where all possible  $x \in \mathcal{X}$  correspond to the same cost value  $y$ . Besides, *most* real instances have the property that *most* points  $x \in \mathcal{X}$  that are close to each other have corresponding points  $y \in \mathcal{Y}$  close to each other as well. The problems we humans are interested in have some internal structure and some degree of predictability or continuity about them.

Intuitively, the NFL theorem illustrates that if knowledge of the problem is not incorporated into the algorithm, then there are no formal assurances that the algorithm will be effective. Rather, effective optimization relies on a fortuitous matching between the problem and the algorithm. It is very important to incorporate problem-specific knowledge into the behavior of the algorithm. Based on the NFL theorem emerges the motivation to find a set of heuristics that jointly (and complementarily) can perform well among a wide range of realistic and interesting possible instances. Poli and Graff [115] argue why in most realistic fitness functions there may be a free lunch for computer scientists developing hyper-heuristics.

Representation is a simplification of an instance state needed in several existing evolutionary models for hyper-heuristic construction [128, 138, 139, 141, 143]. In those models, the numerical representation is applied for *complete instances* to be solved (in BPP: where no piece has been placed yet) as well as for *instances partially solved* (in BPP: where some pieces have already been placed). For a given problem instance, a numerical representation is computed in every intermediate state until it is completely solved (see Figure 1.3). Not all features related to a problem can be taken into account. From the selection of these features relies highly the performance of the hyper-heuristic, since the problem state is related with the selection of single heuristics. In those models where an instance state representation has

been required, feature selection has been done with domain knowledge of the particular problem. It is required to have a more general method for establishing an adequate problem state representation.

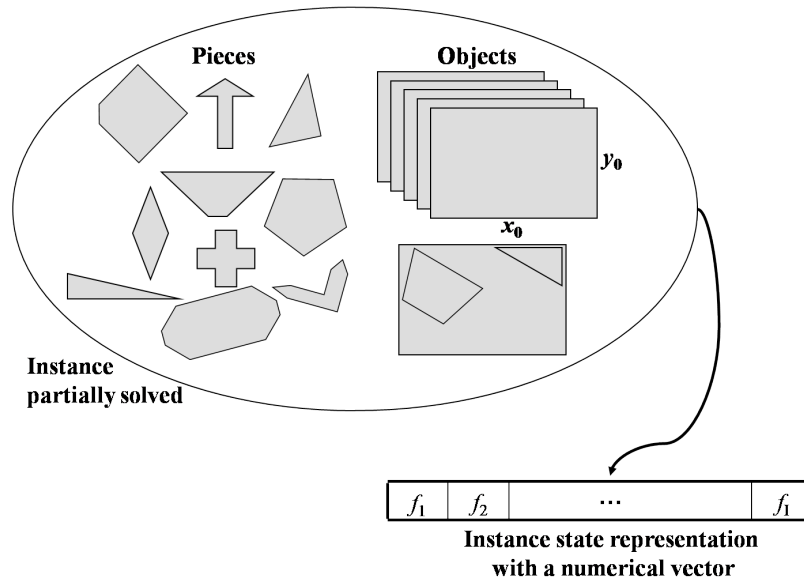


Figure 1.3: Simplified representation of a problem state.

There is also the interest in analyzing the BPP structure. The BPP structure regards those features that are able to characterize particular instances. We want to discover the BPP structure by understanding those features that have a high impact in approximation algorithm performance. Some feature values may influence some heuristics to be better than others. Similar analysis have been done for other optimization problems, such as the Quadratic Assignment Problem [134], the Early/Tardy Machine Scheduling Problem [135] and the Traveling Salesperson Problem [85], but never for the Bin Packing Problem, as far as we know.

### 1.3 Hypothesis and Research Questions

The work is conducted under the hypothesis that utilizing hyper-heuristics is a way of increasing the level of generality of a solution model. This is because utilizing hyper-heuristics for the 2D irregular BPP will increase the range of problems in which a good solution can be obtained. Also, there is the hypothesis that improving the representation scheme would improve the hyper-heuristic performance. Furthermore, modifying (even reducing) the set of heuristics employed in the model, the hyper-heuristics quality would be better. Only this way, combining low-level heuristics into a hyper-heuristic would effectively take advantage of each heuristic strength. We hypothesize that the number, nature and interaction of the low-level heuristics do have an effect in the hyper-heuristic solution quality. We also hypothesize that understanding better the BPP structure is important in a practical way. Knowledge about which feature values have higher impact in which heuristics may be useful in the development of new heuristic-combination techniques.

That is why, the following research questions are generated:

- Is it possible to improve the execution in the 2D BPP of the single heuristic called Djang and Finch, even though it was initially designed for the 1D case?
- Is it possible to develop a general methodology for finding significant features for the 2D irregular BPP without domain knowledge?
- Which are the most important features of the 2D irregular BPP that characterize every instance in such a way that it is possible to determine which is the best heuristic for solving the instance?
- How is the performance of a hyper-heuristic solution model when the most significant features are taken into account in the representation of the problem state?
- Is it possible to generalize a hyper-heuristic model to a wider range of instances including those with non-convex polygons?
- Is it possible to generalize the hyper-heuristic model for other variants of the Bin Packing Problem (for example, 1D)?
- What sort of changes can be made in the set of selection and placement heuristics that would enable the approach to work well for non-convex polygons too?
- How is the performance of a hyper-heuristic solution model when tackling the 2D irregular BPP (non-convex)?
- How are the different heuristics used in the hyper-heuristic, in terms of frequency, sequence of heuristics and the time each heuristic is used before changing to another?
- Is it possible to obtain BPP domain knowledge through a multivariate data analysis such as the Principal Component Analysis?
- Could we find some particular BPP features that make some heuristic more suitable than others?

These questions guided the conducted research as answers to them were found and provided. Worth pointing out is that the list is not in the order answers were found.

## 1.4 Solution Overview

The general solution model is depicted in Figure 1.4. The hyper-heuristic generation framework (rectangle in the middle) receives a set of single heuristics and a set of problem instances as main input. After a training process with a genetic algorithm, our framework produces one or several hyper-heuristics which consist of a set of rules that relate different instances states with the better single heuristic to apply in every case. This dissertation works in four different parts of this general framework (rectangles of text with font in *italics* in Figure 1.4).

First, instances are characterized by a numerical vector before going into the training process of the solution model. Regarding this part of the general scheme, this investigation develops a robust methodology to represent instances employing a numerical vector, leaving



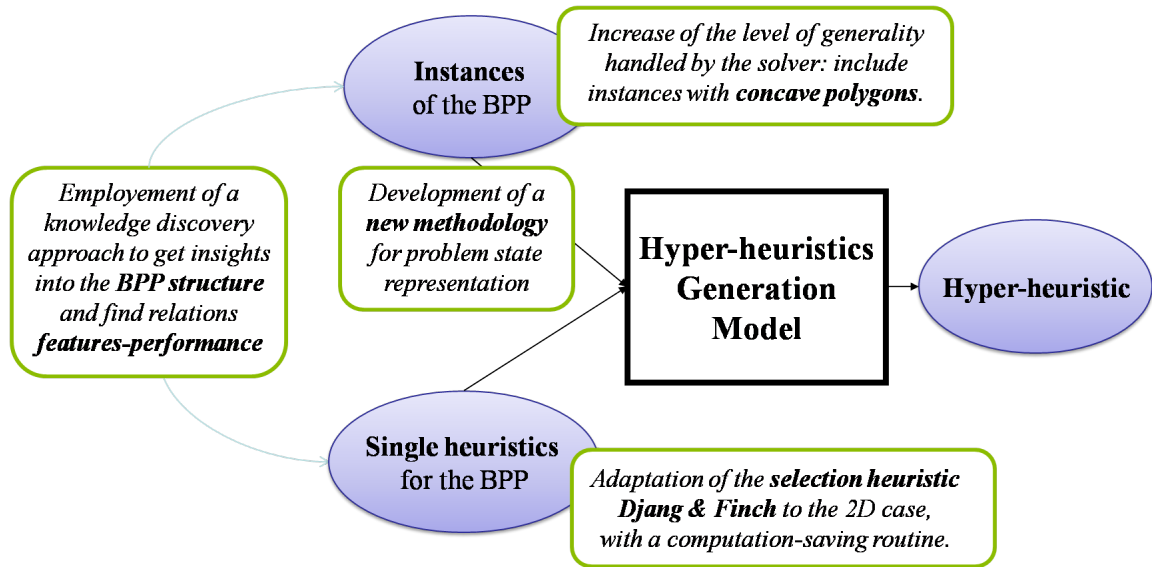


Figure 1.4: General structure of the solution model studied. Fonts in *italics* explain the main improvements to the solution model.

behind intuitive ways for selecting the most relevant features (Chapter 6). The main idea consists in clustering the available instances according to heuristic performance. Then we find which features may predict instances clusters. For clustering, the  $k$ -means algorithm was employed while the Multinomial Logistic Regression technique was used for determining which features can predict instance cluster. Multinomial Logistic Regression is used to predict a categorical dependent variable (cluster), given a set of independent variables (features). The most significant features in the Multinomial Logistic Regression model are as the most related with heuristic performance. The proposed methodology was tested with the 2D irregular BPP with convex polygons. The GA-based process for building hyper-heuristics was run with two different representation schemes concluding that our developed representation shows a significant improvement in performance with respect to a more conventional representation.

The second element from the framework that was analyzed was the heuristic repository. Single heuristics are studied and one selection heuristic (Djang and Finch (DJD)) is selected for further analysis and improvement. The DJD heuristic was originally designed for the 1D BPP. In the two-dimensional case, not only is it the case that the piece's size is important but its shape also has a significant influence. Therefore, DJD as a selection heuristic has to be paired with a placement heuristic to completely construct a solution to the underlying packing problem. A successful adaptation of the DJD requires a routine to reduce computational costs, which is also proposed and successfully tested in this paper. Results, on a wide variety of instance types with convex polygons, are found to be significantly better than those produced by more conventional selection heuristics (Chapter 7).

Still working with the single heuristics part of the model, the number of single heuristics in the heuristic repository was reduced dramatically from 40 (in the first implementation, Chapter 5) to only six. This was done after discarding those heuristics that were not the best in any of our testbed instances. Having only good heuristics in the repository, the speed in which the genetic algorithm converges to a good hyper-heuristic increases. Also, in order to

improve our understanding of the single heuristics role when employing hyper-heuristics, an analysis of the interaction among single heuristics in hyper-heuristic solutions is performed. We found some interesting patterns about heuristic alternation when solving instances with hyper-heuristics. These findings give us interesting ideas for further improve the solution model. Statistical analysis is included to support our study (Sections 8.3.2 and 8.3.4).

The third aspect to be improved is regarding the kind of instances that the produced hyper-heuristics can solve. The hyper-heuristic framework is tried with a more general testbed of instances that goes from 1D to 2D instances with convex and non-convex polygons (Chapter 8). The idea is to find hyper-heuristics in the form of rules that automatically choose the best heuristic to apply to a given instance of any kind. In the past, these problems had been tackled separately by using a variety of methods and techniques because they exhibit different properties. With our framework, once a hyper-heuristic has been evolved using a training set of instances, it can be reused on any new instance of any kind producing competitive results, and comparable against those provided by the best single heuristic per problem.

Finally, a knowledge discovery approach is used to reveal the problem features or combination of features that influence the performance of bin packing heuristics and hyper-heuristics. Our instance testbed includes 1417 instances for this part of the dissertation. Using the Principal Component Analysis method, the problem instances, characterized by 9 features, are visualized in two dimensions. These 9 features were selected from a larger set of 23 by the proposed methodology that selects the subset of features that are more strongly related with algorithm performance. Different combinations of features characterize instances in each section of the 2D graph produced by the Principal Component Analysis method, and heuristic performance is over imposed over the 2D graph. This visualization approach reveals new knowledge on the relationship between bin packing problem features and heuristic and hyper-heuristic performance (Chapter 9).

## 1.5 Main Contributions

The main contribution of this work is to build a more robust and reliable framework to generate hyper-heuristics that intelligently combine single heuristics to solve a wide range of instances of the Bin Packing Problem. By doing this, the following are the given contributions:

- The design and implementation of an algorithm that produces random problem instances with irregular convex polygons. This algorithm requires several user-defined parameters to control the irregularity of the produced random instances.
- An implementation of the proposed evolutionary framework to produce hyper-heuristics for solving the 2D irregular BPP (convex). The analysis of the results of this first implementation produced several suggestions for improving the framework.
- A robust six-step methodology to find the most relevant features of a problem that are most related with heuristic performance. The features selected through this methodology will represent each problem instance state in the genetic algorithm process.
- A new way to represent an instance of the BPP in a numerical vector. Expertise is not required to establish this new representation scheme.

- A design of a time-saving routine for the DJD heuristic, making this heuristic a fast and effective algorithm for selecting the next pieces to be placed.
- An analysis of two important parameters of the DJD heuristic: the initial level of fullness and the allowed waste incremental; finding those values where the DJD heuristic produces the best results.
- A review and improvement of the parameters employed in the genetic algorithm of the solution model.
- The design and implementation of an algorithm that cuts a convex polygon into two new shapes, one of which is a concave polygon. This algorithm is the basis for generating random instances with concave polygons.
- An implementation of geometric functions required to deal with concavities.
- A solution framework able to generate hyper-heuristic for solving at the same time 1D and 2D instances of the Bin Packing Problem.
- A deeper understanding about how the single heuristics interact in a hyper-heuristic solution. We analyzed how the different single heuristics alternate when hyper-heuristics solved different kinds of instances. The number of times that a given single heuristic is applied before changing to another is analyzed as well.
- A deeper understanding of the BPP structure based on the Principal Component Analysis technique as a graphic tool. We visualize in two dimensions our instances tested which are characterized by a larger number of variables. This analysis finds relationships between features and heuristic/hyper-heuristic performance. We expect that this new knowledge will help in the design of new solution approaches for the BPP.
- A large part of the code programmed for this research was used further in a development of a hyper-heuristic solution model for a multiobjective 2D Bin Packing Problem [64].

Along the development of this dissertation, there has been a set of published papers that has also shown the contributions that this research has provided to the scientific community. The complete list is presented in Appendix A.

## 1.6 Scope and Limitations

The proposed solution tackles the problem delimited in the problem statement section. Conclusions from this work are related with the Bin Packing Problem of one and two dimensions. The Strip Packing Problem, another popular version of the C&P problem, is not touched in this dissertation. One-dimensional problem instances were drawn from the literature and present a variety of characteristics. Two-dimensional problem instances were built artificially because of the lack of benchmark problems in the literature; nevertheless, these instances have a wide range of characteristics as well. Allowed rotation for pieces is 90, 180 and 270 degrees. All problem instances have identical object sizes and all 2D objects are squares. It is worth

noting that 2D instances does not allow curves or holes. We do not consider concave shapes with more than one internal angle larger than 180 degrees. So, pieces with an *entrance* to an interior area that fit exactly with pieces with a protuberance (see Figure 2.11) are discarded from this investigation. As a consequence of this, all positions in the optimal solutions can be achieved using sliding operations.

All single heuristics presented in this work are constructive and single-pass, which means that they are algorithms that build solutions from scratch placing pieces one after another and never revising a piece already placed. This kind of heuristics has the advantage of being very fast, but their performance would be unfairly compared with (meta-)heuristics that consider many positions before giving a solution. The idea is to build a hyper-heuristic that successfully combine single heuristics to produce good solutions in only one pass.

Part of the available code was previously employed in the investigation reported in [52] and [138]. The framework was done with the JAVA programming language.

## 1.7 Thesis Organization

This dissertation is organized in nine chapters. The following chapters present all the technical details that are relevant to this research.

Chapter 2 talks about the state of the art regarding the tackled problem and presents some related background relevant to the research, including topics about computational complexity, geometry and hyper-heuristic approaches. Then, Chapter 3 explains in detail the solution model this dissertation works with. Chapter 4 talks about the research methodology and the experimental setup. It shows the main steps followed during the research process and presents the problem instances in which the proposed contributions are tested. The testing instances are described as well as the algorithm employed to randomly create 2D instances with some concave pieces. Finally, some algorithms implemented for dealing with geometry are presented in this chapter. In Chapter 5 a first experimentation employing the solution model is explained. Chapter 6 focuses in the problem-state representation and develops a novel way of selecting relevant features of a given problem. Chapter 7 develops the adaptation to the 2D BPP of the DJD heuristic. Chapter 8 extends the solution model making it capable of generating hyper-heuristics for 1D and 2D instances, including instances with non-convex polygons. This chapter integrates improvements to the single heuristics repository and the problem state representation scheme from previous chapters ensuring better quality hyper-heuristics. Chapter 9 presents 2D maps where many instances are plotted to visually assess differences and similarities among instances. By superimposing heuristic performance we are able to produce insights into the BPP structure. Finally, Chapter 10 provides some conclusions related to the work that was carried out throughout this research project. Some future guidelines and challenges are presented along with some possible ways to tackle them and improving obtained results and performance.

# Chapter 2

## Related Background

The Bin Packing Problem is a particular case of the more general C&P problem. In this chapter the C&P problem is described with emphasis in the two dimensional case. Although, some review is devoted to the one-dimensional case, because this problem is solved in the framework described in Chapter 8. The C&P problem has been theoretically tackled in the last decades and the main approaches utilized for the 2D version are discussed showing that the most recent approach, called hyper-heuristic, had not been applied to the 2D irregular BPP (non-convex) until this investigation. This is a wide problem, with many variations, that is why several typologies have been built.

The C&P problem is reviewed in Section 2.1 in which two different typologies are presented. Then, Section 2.2 shows the importance of studying the cutting problem because of its numerous applications. Section 2.3 discusses the complexity of the problem. Some issues about geometry are addressed in Section 2.4 while Sections 2.5, 2.6 and 2.7 talk about some approaches for solving the 2D C&P problem found in the literature. Section 2.8 presents the basics about the hyper-heuristic search approach; Section 2.9 goes deeper in how hyper-heuristics has been applied to the 2D C&P problem and Section 2.10 shows some examples of hyper-heuristics implemented in other combinatorial problems.

### 2.1 Cutting and Packing (C&P) Problems

With only a couple of exceptions, research in Cutting and Packing (C&P) problems started after the middle of the twentieth century. According to Haessler in 1991 [66], the earliest work appeared in the paper industry with the first known formulation stated, in 1939, by the Russian economist Kantorovich [86], whose paper was translated into English in 1960. In 1940, Brooks et al. [19] wrote a mathematical paper about the dissection of rectangles into squares. Another one of the earliest publications in the topic was from Gilmore and Gomory [57], who introduced, in 1961, a delayed pattern generation technique for solving a one dimensional cutting problem using linear programming. Since then, scientific work has grown rapidly through several disciplines. The same logic structure of the problem appears in areas such as Computer Science, Mathematics, Logistics, Operational Research and Engineering. That is why, there has been used a variety of terms according to specific applications. For example:

- *Parts nesting* in the ship-building industry [44].
- *Marker layout problem* in the garment industry [44].
- *Memory allocation, Multiprocessor scheduling and Compaction problem* [65] in the computer industry.
- *Floor planning* in the VLSI (very-large-scale integration) industry [121].
- *Coil Slitting* [49].
- *Container loading* which is a three-dimensional problem [53] and it is similar to *Pallet loading*, though in practical applications the two variants of the loading problem can be distinguished by their constraints [74].
- *Capital Budgeting and Change Making* [49].
- *Scheduling* where dealing with a one-dimensional set of objects (*jobs*) that need to be assigned to a finite set of containers (*machines*).

Jacquenot et al. [79] talks about a higher problem category: **placement problems** which gather C&P problems as well as the layout problems. In a C&P problem, components are only geometrically related to each other, whereas in a layout problem, components are geometrically and functionally related to each other.

Also, according to different variations or characteristics of the Cutting and Packing problem, many terms have been used. For example, *bin packing* when there are several objects in which a number of parts are to be packed; *strip packing* in the case in which a number of parts are to be cut from an object of indefinite length. *Knapsack problem* is usually related to the one dimensional problem. The *assortment problem* addresses the issue of choosing proper dimensions for the large objects. Wäscher et al. [154] did an important effort of unifying all these terms, clarifying the characteristics to apply to each (see section 2.1.2).

Due to the extensive work done in this and related problems, several typologies have been proposed. Dyckhoff's typology [48] was the one which ruled since 1990. In 2006 Wäscher et al. [154] suggested a more complete one. These typologies are described in sections 2.1.1 and 2.1.2.

### 2.1.1 Dyckhoff's Typology

Because C&P problems are found in many different application areas, Dyckhoff in 1990 [48] built a systematic and comprehensive classification for an adequate exchange of solution approaches across disciplines. His purpose was to unify the different names and the different use of notions in the literature and to concentrate further research on special types of problems.

The Dyckhoff's classification integrates a system of 96 types of cutting and stock problems according to four main features and their subtypes as follows:

1. Dimensionality: One (1), Two (2), Three (3) or  $n$  (N).

2. The task: Either (B) use as many small figures as possible to fill all the large objects; or (V) use as few of the large objects as possible to contain all of the small figures (B stands for the German ‘Beladeproblem’ and V for the German ‘Verladeproblem’).
3. Assortment of large objects: One object (O), identical shapes (I) or different shapes (D).
4. Assortment of small figures: Few figures of different shapes (P), many figures of different shapes (M), many figures of few of different and incongruent shapes (R) or congruent shapes (for example, rectangles) (C).

For example, 3/B/O/P denotes all three-dimensional C&P problems where one large object has to be packed with a selection out of a few small different items.

### 2.1.2 Wäscher’s Typology

Although Dyckhoff’s typology initially provided an excellent instrument for the organization and categorization of existing and new literature; over the years some deficiencies of this typology became evident, which created problems in dealing with recent developments and prevented it from being accepted more generally. That is why Wäscher et al. thought about an improved typology [154], which is partially based on Dyckhoff’s original ideas.

Problem types can generally be defined as elementary types or combined types. Five criteria will be used here for the definition of combined problem types of C&P problems:

1. *Dimensionality.* The criterion of the number of the problem relevant dimensions is adopted directly from Dyckhoff’s typology. But Wäscher et al. consider *one*, *two* and *three* dimensions only; because in the literature, occasionally, problems with more than three geometric dimensions are considered (e.g., Lins et al. in 2002 [97]). Problems of this type ( $n > 3$ ) are looked upon as variants, here.
2. *Kind of assignment.* This criterion has proved to be useful in the past. Wäscher’s et al. kept it avoiding German notation. They used *input minimization* and *output maximization* for types V and B respectively.
3. *Assortment of small items.* All small pieces are identical in size and shape (*identical small items*); the small items can be grouped into relatively few classes (in relation to the total number of items), for which the items are identical with respect to shape and size (*weakly heterogeneous assortment*); or only very few elements are of identical shape and size (*strongly heterogeneous assortment*).
4. *Assortment of large objects.* With respect to the assortment of the large objects there are introduced two cases:  
*One large object.* In this case the set of large objects consists of a single element which can be further classified according to its dimensions:
  - (a) *All dimensions fixed.*
  - (b) *One or more variable dimensions.*

And *several large objects* with all dimensions fixed:

- (a) *Identical large objects.*
  - (b) *Weakly heterogeneous assortment of large objects.*
  - (c) *Strongly heterogeneous assortment of large objects.*
5. *Shape of the small items.* In the case of two- and three-dimensional problems: *Regular small items* (rectangles, circles, boxes, cylinders, balls, etc.) and *Irregular* (also called: *non-regular*). Bennell and Oliveira [12] define a piece as irregular if it requires a minimum of three parameters to identify it. For example, a circle needs just a single parameter, the radius, and a rectangle needs two parameters, its length and width.

The two criteria *kind of assignment* and *assortment of small items* will be used in combination in order to define six *basic problem types*. For each basic problem type, the subsequent application of the criterion *assortment of large objects* will provide *intermediate problem types*. Figure 2.1 shows the names of the basic and intermediate problem types. Note that basic types *Identical Item Packing Problem* and *Open Dimension Problem* are not further differentiated. Further characterization of intermediate problem types with respect to the number of problem relevant dimensions (*dimensionality*) and –in the case of problems of two and more dimensions– with respect to the *shape of small items* will provide *refined (combined) problem types*. The resulting subcategories are characterized by adjectives which are added to (the names of) the intermediate problem types (IPT) according to the following system: 1, 2, 3-dimensional {rectangular, circular, ..., irregular} IPT.

A problem instance which only exhibits the defining properties, but no additional constraints or characteristics could be interpreted as being (an instance of) a (*first-level*) *standard problem (type)*. *First-level non-standard problems (problem types)* are characterized by the properties defining the respective problem category and additional constraints and/or characteristics. *Second-level standard problems* are those first-level non-standard problems considered as well known standard problems for the scientific community.

The (two-dimensional) *Strip Packing Problem* is an Open Dimension Problem in which a set of two dimensional small items has to be laid out on a rectangular large object; the width of the large object is fixed, its length is variable and has to be minimized.

According to this typology proposed by Wäscher et al. [154], the target problem of the present dissertation falls into the categories of two-dimensional, input (value) minimization, weakly heterogeneous assortment of small items, identical large objects, and irregular (polygonal) small items. The problem is considered as **2-dimensional irregular Single Bin Size Bin Packing Problem (2D irregular SBSBPP)**, see Figure 2.1). In this document this same problem is been denoted as 2D irregular BPP. The problem is offline, and therefore the list of pieces to be packed is static and given in advance.

## 2.2 Industrial Applications for the C&P Problem

The reduction of production cost is one of the major issues in manufacturing industries. High material utilization is of particular interest to industries with mass-production, since small



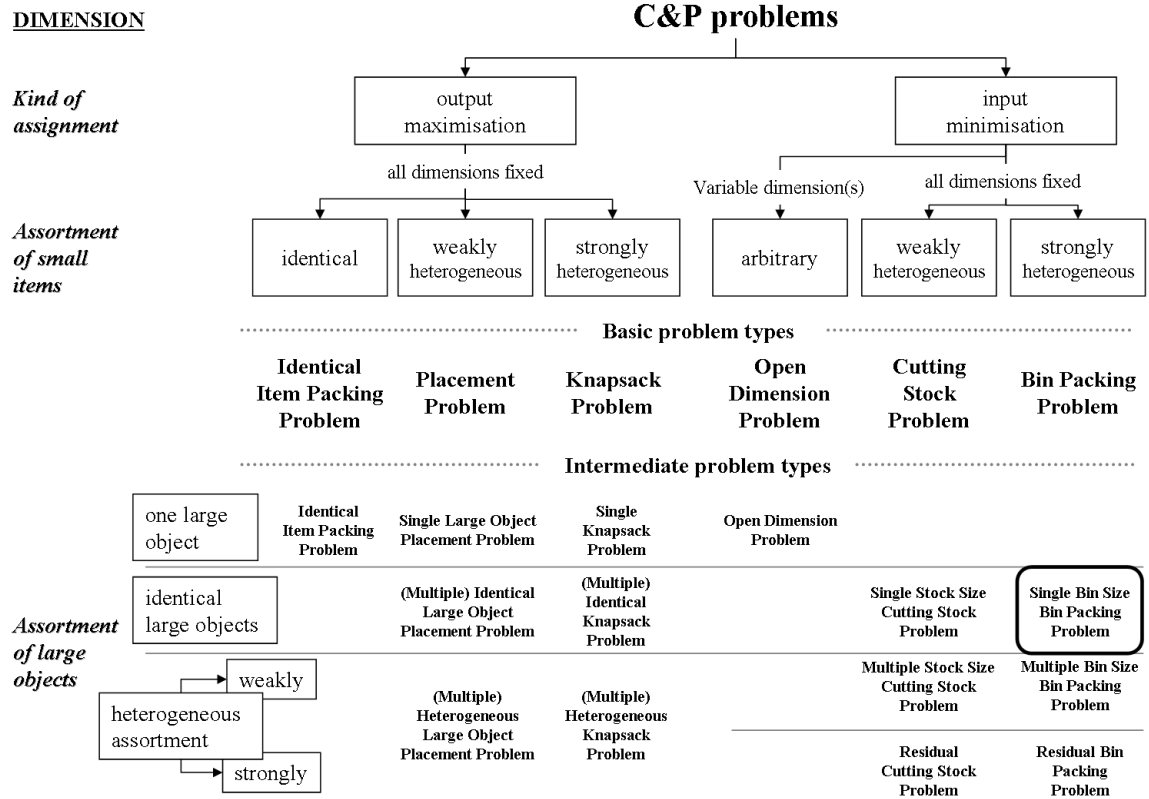


Figure 2.1: Basic and intermediate problem types for the C&P problem in Wäscher’s Typology [154].

improvements of the layout can result in large savings of material and considerably reduce production cost [72, 77]. For example, in the garment industry, many pieces are cut using the same layout. For pants manufacturing, each time a layout is used, sixty layers of cloth are cut at the same time [95].

The following are the main industrial applications found in literature for the 2D problem:

- *Metal industry.* The sheet metal industry has to deal with regular as well as irregular nesting problems [72]. In the steel industry, for example, the small items are called product reels while the large objects are called raw reels. Usually raw reel widths are considered to be fixed while length dimension is variable [87]. For profiling (sheet metal cutting) within the steel-cutting industry, it is imperative that we are able to handle arcs, concavities, and holes [23].
- *Shipbuilding industry.* Plate parts with free-form shapes for use in the inner frameworks of ships are cut from rectangular steel plates [109].
- *Garment industry.* The textile industry usually utilizes coiled material and hence is mainly concerned with the strip packing problem [72]. In this industry, pieces usually are concave, may contain arcs and may have high multiplicity (a small number of different pieces, many times each).

- *Shoe industry.* In this industry both, the object (natural leather) and the pieces are irregular and mostly concave [40].
- *Lumber industry.* Here, most pieces to be cut are rectangles for making furniture [36]. The cutting task in the lumber industry is a two and a three dimensional problem [82].
- *Glass industry.* In this material, it is common the need to cut rectangles [27].
- *Kiln planning* for the ceramic industry. Capacity planning of tunnel kiln in ceramic and porcelain industry has become more interesting as the energy saving issues is growing concern of operations management. Here, pieces are mostly circles (i.e. plates, dishes) with equal or different diameters [106].
- *VLSI design.* Very-large-scale integration (VLSI) is the process of creating integrated circuits by combining thousands of transistor-based circuits into a single chip. One of the problems related VLSI systems is its physical design which includes solving the *compaction problem*. The goal is to minimize the area of the layout, while preserving the design rules and not altering the function of the circuit [65].
- *Reconstruction of strip shredded text documents (RSSTD)* which is of great interest in investigative sciences and forensics [118]. Techniques for the 2D irregular BPP could be used in reconstruction of manually torn paper documents, where shape information can be exploited to some degree.

Some of the most important applications of the C&P problem in one dimension include: cutting linear elements like steel rod or marble shelf from standard lengths [91], vehicle loading and scheduling (assign jobs of different lengths to machines) [56].

### 2.2.1 Industrial Constraints

Due to practical considerations, different industries work with different constraints and objectives. Although the main objective is trim minimizing; in some cases, if there is a large area of trim, this can be reused. That is, there are cases where not only the amount but also the distribution of the trim are important. Other consideration can be the prioritization of pieces or precedence constraints. The following are some constraints in the two dimensional problem specifically related to some industries:

- In the sheet metal industry, the non-homogeneous properties of metal such as grain orientation limit the number of possible orientations of the items. If bending operations follow, the parts can only be rotated at a specified angle [74].
- In the garment industry, the grain and pattern of the fabric means that the orientation of the pieces is fixed (or a 180 degrees rotation is allowed). It is usual to have a few different pieces to be cut many times. Since fabric is thin, many stock sheet are put one over another and they are cut together to save cutting time. In many cases it may not be possible to mirror the parts as the fabric has different properties at the other side [72].

- The manufacture of leather shoes involves ensuring that the direction of certain components lies along lines of tightness and assigning parts to graded quality regions. Region grading conforms to a grading system commonly used in shoemaking, which ranges from grade 1 to 5. Leather is a natural material and can show variations in strength and flexibility depending on direction [40]. There is also a problem caused by imperfect or unusable areas on the stock-sheet.
- In the lumber cutting industry orientation may be important due to the grain of the wood [44]. Also, rectangles must be cut from stock sheets using guillotine or edge-to-edge cuts made parallel to the edges of the stock sheet [107]. Guillotunable means that the parts can be obtained by straight cut through the remaining layout only.

In spite of these differences all these problems have the common requirement of finding a feasible layout of the pieces on the stock-sheet.

For the one-dimensional case (i.e. *Scheduling*) precedence constraints are not uncommon [53]. When the one-dimensional Bin Packing Problem is applied to loading vehicles a conflict may occur where some (flammable, explosive or toxic substances cannot be placed in the same vehicle [56].

## 2.3 Packing and NP-Hardness

One of the most important conventions in complexity theory is the definition of problem classes. Combinatorial problems can be stated as decision problems where a solution corresponds to a correct *yes* or *no* response. An optimization problem is where some value needs to be minimized or maximized. According to the algorithm complexity, the following problem classes can be distinguished:

- **Class P.** A problem is assigned to the **P** (polynomial time) class if there exists at least one algorithm to solve that problem, such that the number of steps of the algorithm is bounded by a polynomial in  $n$ , where  $n$  is the length of the input. There is a wide agreement that a problem has not been *well-solved* until a polynomial time algorithm is known for it [55]. The membership in this class means, theoretically, that even large problem instances can be solved with exact routines. Increases in problem size normally have a small impact on the computation time. However, the solution of large problems might become impractical using conventional computer systems. Thus, it may be still worthwhile to search for approximation algorithms that are of lower polynomial order.
- **Class NP.** A problem is in **NP** (nondeterministic polynomial time) class if there is a 'guess-and-check' algorithm for suggested solutions. The nondeterministic phase consists in *guessing* a solution. Following, there is a deterministic phase which verify the solution proposed. To verify a solution consists in checking if it satisfies the problem criteria. Although problems in this class are *easy* to verify, they are not *easy* to solve. By this definition, the class NP contains the class P. The nondeterministic phase for a problem in class P would do nothing, and then, in the deterministic phase a polynomial algorithm would solve it. So, for problems in class NP, it is possible to create an algorithm then *invents* a proposed solution, verifies it. We expect that doing this a large

number of times would end in a *good* solution [9]. A problem  $P$  reduces to a problem  $Q$  if any algorithm that solves  $Q$  also solves problem  $P$  provided an appropriate conversion is found.  $T$  is a polynomial reduction or transformation from  $P$  to  $Q$  if and only if: a)  $T$  is executed in time polynomially bounded; b) for every instance  $x$  of  $P$ ,  $T(x)$  is an instance of  $Q$ ; and, c) for every instance  $T(x)$  of  $Q$ ,  $x$  is an instance of  $P$ . Polynomial reductions are the most important ones in this context. A problem  $P$  polynomially reduces to  $Q$ , if a polynomial time algorithm for  $Q$  implies the existence of a polynomial time algorithm for  $P$ .

- **Class NP-hard.** Problems to which all members of NP polynomially reduce are referred to as NP-hard. A NP-hard problem is not always a NP problem. Thus, a polynomial time algorithm for a NP-hard problem, would produce an algorithm for every member of NP at the same time. However, this is not expected to happen since there is a general idea of  $P \neq NP$ . Many optimization problems are known to be NP-hard. A problem is said to be NP-hard in the strong sense (or strongly NP-hard), if it remains so even when all of its numerical parameters are bounded by a polynomial in the length of the input. For the 1D BPP, for example, numerical parameters are the pieces sizes. It is proved that a strongly NP-hard problem cannot be solved by a pseudo-polynomial time algorithm. An algorithm runs in pseudo-polynomial time if its running time is polynomial in the numeric value of the input (which is exponential in the length of the input in terms of its number of digits).
- **Class NP-complete.** Problems that are in class NP-hard and also in class NP are called NP-complete. As all the members of class NP-complete are decision problems, it does not contain optimization problems in the strict sense. Optimization problems are NP-hard, but have analogue decision problems which are NP-complete.

As stated in section 2.1.2, the C&P problem is an optimization problem with many variations. The 1D BPP, for example, is known to be NP-hard, because the decision version of the Bin Packing Problem is a polynomial time transformation (reduction) from the NP-complete problems called PARTITION and 3-PARTITION [55]. The PARTITION problem consists of deciding whether a given set of integers can be partitioned into two disjoint subsets that have the same sum. The 3-PARTITION problem is to decide whether a given set of integers can be partitioned into disjoint triples that all have the same sum denoted by  $B$ . It is a restriction that the size of each element of the given set is strictly between  $B/4$  and  $B/2$ . Reduction of the 1D BPP from the PARTITION problem can be explained as follows: Given items of size  $a_1, \dots, a_n$ , make an instance of Bin Packing with items of the same size and bins of size  $\sum a_i/2$ . There is a solution for Bin Packing that uses 2 bins if and only if there is a solution for the PARTITION problem.

In the case of the two-dimensional variations of the C&P problem, the open dimension problem is NP-hard even for rectangular pieces. The proof consists of reducing the PARTITION problem to the regular open dimension problem: for an instance  $(a_1, a_2, \dots, a_n)$  of PARTITION, we build rectangular pieces of height 1 and width  $a_i$  ( $i = 1, \dots, n$ ). We put the blocks in a container of height 2. The length is going to be minimized. PARTITION has a solution if and only if the blocks can be compacted to the length  $\sum a_i/2$  [95]. Another variation of the C&P is the two-dimensional identical item packing problem (for rectangles),

which is claimed to be NP-hard although it has not been proved [16]. Besides, geometrical complexity is introduced when considering convex and non-convex polygons and it has been shown that even compacting to a local optimum can require an exponential number of moves [96]. For the irregular case, as far as we know, there are neither algorithms with known worst case behavior bounds nor algorithms computing optimal solution in any way. In practice, only experimental evaluation and comparison is possible [17].

A problem is referred as *intractable* if it is so hard that no polynomial time algorithm can possibly solve it [55], that is why, the C&P problem is, in general, intractable. Moreover, majority of the C&P problem variations are strongly NP-hard, thus, unlikely to admit pseudopolynomial-time algorithms [17].

## 2.4 The Geometry for Placing Irregular Polygons

Despite its relevance to industry, research publications regarding irregular pieces are relatively low when compared to other cutting and packing problems. One explanation offered is the perceived difficulty and substantial time investment of developing a geometric tool box to assess computer generated solutions [11]. In this section, there are described the main geometric methodologies employed in packing irregular polygons. Determining which is the most appropriate approach to implement is not just a matter of how well they perform, but also how difficult they are to implement robustly.

### 2.4.1 Pixel/raster Method

Raster methods are approaches that divide the continuous stock sheet into discrete areas, hence reducing the geometric information to coding data by a grid represented by a matrix. The simplest coding scheme uses the value of 1 to code the existence of a piece, and the value of 0 to denote an empty space. Other coding schemes includes different values to the frontier and the interior of the pieces [11]. Advantages of this method is that calculating distances and feasibility is easy. Also, this method of representing pieces can manage non-convex and complex pieces as easily as simple polygons. However, disadvantages are that the method is memory intensive and cannot represent exactly pieces with non-orthogonal edges.

### 2.4.2 Direct Trigonometry

When representing the pieces as polygons the amount of information is proportional to the number of vertices and does not depend on the absolute size of the pieces. There exist well known tests for line intersection and point inclusion.

For line intersection, the  $D$ -function gives the relative position of a point  $P$  with respect to an oriented edge  $AB$  (see Figure 2.2). The  $D$ -function can be defined as follows:

$$D_{ABP} = (X_A - X_B)(Y_A - Y_P) - (Y_A - Y_B)(X_A - X_P) \quad (2.1)$$

Depending if  $D_{ABP}$  is negative or positive, the point  $P$  is on the left or the right side of the edge  $AB$ . The definition of left and right is as follows: if an observer would stand at point  $A$  looking in the direction of  $B$ , point  $P$  would be at the observer's left or right. If  $D_{ABP} = 0$ ,

the point  $P$  is on the supporting line of edge  $AB$ . This way, the  $D$ -function is an efficient tool for characterizing the relationship between two edges, and so, for detecting if a pair of polygons intersect [11].

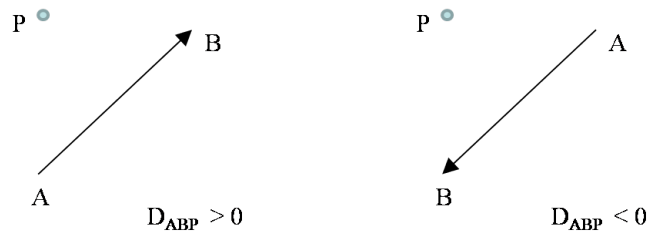


Figure 2.2: Interpretation of the  $D$ -function.

### 2.4.3 The No-Fit Polygon (NFP)

The NFP is a geometry tool that has practically become a prerequisite for solving irregular packing problems; it gives the set of non-overlapping placements for a given pair of polygons [76]. The first application of no-fit polygon techniques within the field of cutting and packing was presented by Art in 1966, although the term *shape envelop* was used [7]. It was ten years later that the term **no-fit polygon** was introduced by Adamowicz and Albano who approached the irregular stock cutting problem by using no-fit polygons to pack shapes together using their minimum enclosing rectangles [2]. The term *configuration space obstacle* is often used to denote the NFP within the field of engineering and robot motion planning but the term has also been used with respect to cutting and packing [24]. Utilizing the no-fit polygon makes it unnecessary to check for overlapping pieces [29]. Whilst the generation of the no-fit polygon is academically challenging, it is a *tool* and not a *solution* [24].

The NFP of two polygons  $A$  and  $B$ , denoted as  $NFP_{AB}$  is the resulting polygon from a sliding operation between  $A$  and  $B$  where each has a specific role within the operation. Both polygons have fixed orientation.  $A$  has a fixed position where the origin is assumed to be at  $(0, 0)$ ,  $B$  is the tracing polygon that moves around the perimeter of  $A$  to perform the sliding operation. The NFP is defined by placing  $B$  in a touching position with  $A$  and marking the locus of a reference point on  $B$  as it traces around the boundary of  $A$ . The tracing motion is performed in such a way that  $A$  and  $B$  always touch, but never overlap. The locus of the reference point forms a closed path that is  $NFP_{AB}$  [11]. Figure 2.3 illustrates an example of  $NFP_{AB}$ . Clearly while  $A$  is fixed at  $(0, 0)$ , if  $B$  is placed so that the reference point is inside  $NFP_{AB}$  then  $A$  and  $B$  overlap and if the reference point is on the boundary then  $A$  and  $B$  touch.

While both polygons are convex, the concept of the NFP and its realization are quite simple. First, the different roles of each polygon are recognized, and as a result the edges of the polygons must have different orientations (Figure 2.4a). We will adopt the convention that polygon  $A$  (fixed polygon) is counterclockwise and polygon  $B$  (orbiting polygon) is clockwise. Second, a slope diagram is built (Figure 2.4b); and finally, the order of the edges of the NFP of two convex polygons is equivalent to sorting the edges of both polygons in slope order (Figure 2.4c). This method results in the shape and orientation of the NFP but not its

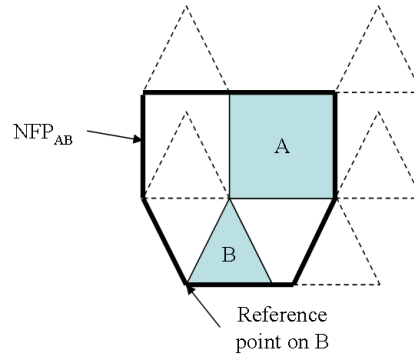


Figure 2.3: Generation of the NFP between a square (fixed polygon) and a triangle (orbiting polygon).

position. In order to use the NFP to determine overlap, an origin from which the position of each polygon is measured must exist. This position is determined with respect reference point of the orbiting polygon.

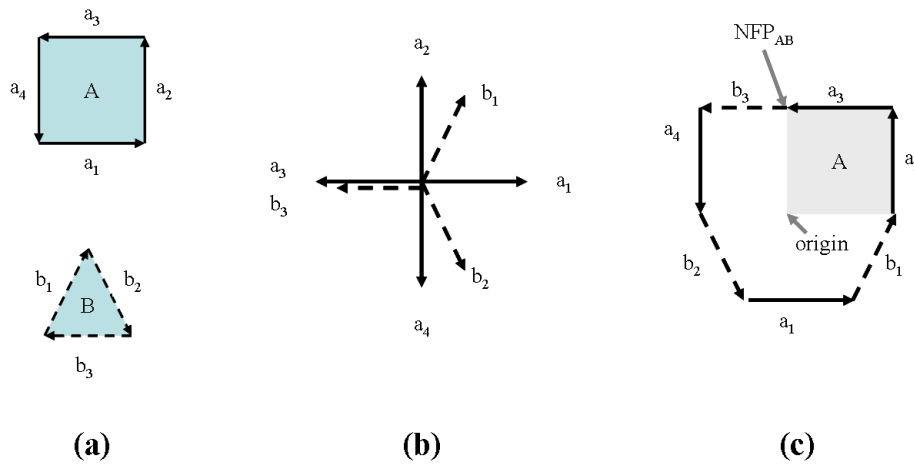


Figure 2.4: (a) The orientation of  $A$  and  $B$ ; (b) the slopes diagram; and (c) the NFP.

For nonconvex polygons, there exist three core approaches in the cutting and packing literature [11]; (a) the orbiting algorithm; (b) Minkowski sums; and (c) decomposition into star shaped polygons or convex polygons. While the no-fit polygon is a powerful geometric technique, there are several issues that limit its scalability for industrial applications. No-fit polygon techniques are notorious for the large quantity of degenerate cases that must be handled to make it completely robust [23]. A significant drawback of this approach is the non-trivial task of developing a robust NFP generator for general nonconvex polygons [12]. However, alternative methodologies have been proposed [13, 24]. Examples of this type of procedure implemented for the strip packing problem are found in [77, 62, 46, 30, 24].

### 2.4.4 The Phi Function

The phi-function is a recent innovation in dealing with the geometric issues for nesting problems [11]. The phi-function for cutting and packing were conceived and applied by Stoyan et al. [136]. It is a mathematical expression that represents the mutual position of two objects. Specifically the value of the phi-function is greater than zero if the objects are separated, equal to zero if their boundaries touch and less than zero if they overlap. When the phi-function is normalized its value is the Euclidean distance between the two objects. Stoyan et al. derive the phi-function for primary objects; these are circles, rectangles, regular polygons, convex polygons and the complement of these shapes. Shapes that are not primary objects can be represented as a union or intersection of the primary objects. As far as we know, there does not exist a robust algorithmic procedure for generating the phi-function for irregular pieces. Therefore, this is a promising research area. It is expected that the use of the phi-function will be spread as long as more powerful tools are developed [11].

## 2.5 Exact Solutions for the 2D C&P Problem

A variety of approaches have been done. For relatively small instances of the 2D C&P problem, exact algorithms can be applied in a reasonable time.

### 2.5.1 Linear Programming

As far as we know, the only method optimal in some sense for the irregular case has been proposed by Adamowicz [1]. Solution is obtained by iterative application of a two-stage procedure. The first is a linear programming problem; its solution minimize linear objective function subject to linear constraints. The second, geometrical stage, checks if the set of elements can be allocated feasibly satisfying geometrical constraints. If the solution does not exist, then new linear constraints resulting from the information obtained in the second step are generated for the new iteration. There are geometrical constraints of two types: absolute, that bound locations on the stock area; and relative, defined in relation to the other elements. This procedure searches in the space of all possible locations of elements maximizing number of elements allocated.

However, this approach is so complex that experimental program is either not completely usable or implements very simplified version of the method [17]. The computational complexity of the geometrical phase grows exponentially in the number of objects and orientations due to the search of candidate allocations, thus this method is rather a theoretical one [17].

### 2.5.2 Tree-Search Procedures

According to Hifi [68] *branch-and-bound* is a well-known technique for solving combinatorial search problems. Its basic scheme is to reduce the problem search space by dynamically pruning unsearched areas which cannot yield better results than already found. The branch-and-bound method searches a finite space  $S$ , implicitly given as a set, in order to find one state  $s^* \in S$  which is optimal for a given objective function  $f$ . Generally, this approach proceeds



by developing a tree in which each node represents a part of the state space  $S$ . The root node represents the entire state space  $S$ . Nodes are branched into new nodes which means that a given part  $S'$  of the state space is further split into a number of subsets, the union of which is equal to  $S'$ . Hence, the optimal solution over  $S'$  is equal to the optimal solution over one of the subsets and the value of the optimal solution over  $S'$  is the minimum (or maximum) of the optima over the subsets. The decomposition process is repeated until the optimal solution over the part of the state space is reached.

With this principle, Viswanathan et al. [150] developed a best-first tree search algorithm based on Wang's [151] bottom-up approach that guarantees optimal solutions for the rectangular two-dimensional placement problem (output maximization). Later, Hifi [68] evolved several exact algorithms for the guillotine rectangular strip cutting & packing problem (input minimization) based on Viswanathan's algorithms.

Algorithms for finding exact solutions for some very particular problem instances are discussed in [89]. These algorithms use the backtracking principle (also called depth-first search). One of the instances solved is the problem of packing 45 identical Y-pentominoes into a square leaving no waste. A pentomino is a polygon composed of five congruent squares. The Y-pentomino is non-convex and is called this way because it resembles vaguely letter Y.

## 2.6 Heuristic Search for the 2D C&P Problem

For many real-world problems, an exhaustive search for solutions is not a practical proposition. The search space may be far too big, or there may not even be a convenient way to enumerate the search space. For example, there may be elaborate constraints that give the space of feasible solutions a very complex shape. The term *heuristic* is sometimes used to refer to a whole search algorithm and is sometimes used to refer to a particular decision process sitting within some repetitive control structure [22]. Many heuristic approaches have been adopted to solve the irregular C&P problem. For example, for irregular shaped parts, previous methods approximate the parts using a bounding rectangle, but this results in material waste [4]. It has always been common to allow configurations with overlapping pieces in the solution space and to penalize these in the evaluation function. However, depending on the severity of the penalty this relaxation results in a tendency to converge to infeasible solutions or to seek out feasible solutions at the expense of overall quality [10].

It is not uncommon that C&P approaches for the Bin Packing variant present at least two phases: first, the selection of the next piece to be placed and the corresponding object to place it; and second, the actual placement of the selected piece in a position inside the object according to given criteria. Some approaches consider a third phase as a local search mechanism. First two approaches are done while *working with partial solutions* because heuristics constructs a layout piece by piece. In the case of a single-pass construction heuristic, the approach can often produce reasonable quality solutions with little computational cost [12]. In addition, feasibility is embedded into the heuristic since each piece is placed in a feasible position on the stock sheet and not moved. A key characteristic of local search approaches is the process of iteratively making small changes to a *complete solution*.

### 2.6.1 Selection Heuristics

Regarding the selection criteria, most researchers have focused upon exploring different ways of finding good permutation of pieces. Okano [109] obtains an ordering of pieces with respect to their areas and the similarities among them. Dowsland et al. [46] use eight static orderings, which have the common strategy of trying to place the difficult-to-place pieces first. Dynamic selection permits all pieces to be available to be placed next [12], for example, Bennell and Song [14] use beam search. This approach searches the breadth first tree, and prunes the tree at each level according to two evaluation functions.

Here are some of the selection heuristics found in the literature for the one and two-dimensional BPP that are implemented through this dissertation.

1. **First Fit (FF).**- Considers the opened objects in turn in a fixed order and place the item in the first one where it fits. This is a very straightforward greedy approximation algorithm. The algorithm processes the items in arbitrary order. For the 1D case, this algorithm achieves an approximation factor of 2. This is due to the observation that at any given time, it is impossible for 2 bins to be at most half full. The reason is that if two bins are at most half full, they would be able to merge. Thus if we have  $B$  bins and optimum value  $OPT$ , therefore  $B \leq 2OPT$  [148]. Recently, Xia and Tan [157] presented improved bounds for this algorithm showing that the absolute performance ratio of FF is at most  $12/7$ .
2. **First Fit Decreasing (FFD).**- Sorts pieces in decreasing order, and the largest one is placed according to FF. It has been shown to use no more than  $11/9OPT + 1$  bins in the 1D case [158].
3. **First Fit Increasing (FFI).**- Sorts pieces in increasing order, and the smallest one is placed according to FF. As FF, this algorithm also achieves an approximation factor of 2 for the 1D case.
4. **Filler.**- Sorts the unplaced pieces in decreasing order and places as many pieces as possible within the open objects. If no single piece could be placed, it opens a new object.
5. **Next Fit (NF).**- Uses the current object to place the next piece, otherwise open a new one and place the piece there.
6. **Next Fit Decreasing (NFD).**- Sorts the pieces in decreasing order, and the largest one is placed according to NF.
7. **Best Fit (BF).**- Places the item in the opened object where it best fits, that is, in the object that leaves minimum waste.
8. **Best Fit Decreasing (BFD).**- Same as the previous one, but sorting the pieces in decreasing order. It has the same worst-case performance than FFD since it uses no more than  $11/9OPT + 1$  bins in the 1D case [158].
9. **Worst Fit (WF).**- Places the item in the opened object where it worst fits (that is, with the largest available room).

10. **Djang and Fitch (DJD).**- Places items in an object, taking items by decreasing size, until the object is at least one-third full. Then, it initializes  $w = 0$ , a variable indicating the allowed waste, and looks for combinations of one, two, or three items producing a waste up to  $w$ . If any combination fails, it increases  $w$  by one twentieth of the object. Chapter 7 deals deeper with this heuristic varying the initial portion of the object that is full before trying combination of pieces.

Some of these heuristics are also described by Ross et al. [128] and Hopper et al. [75].

## 2.6.2 Placement Heuristics

The placement procedure for irregular pieces has attracted more researchers than the exploration of the selection criteria. Once the piece and object are selected, the placement heuristic states the way the piece is located inside the object. Given a piece and an object, two different placement heuristics could arrive to different conclusions about if the piece can or cannot be located inside the object or about the pieces final coordinates.

There are several techniques to generate potential placement positions for the next piece to be placed, and many of them are based on building the no-fit polygon. That is why a description of the NFP is given in section 2.4.3. After generating the possible placement positions, it is necessary to have some criteria for choosing the best position. In the research area of 2D cutting and packing problems the most commonly used method for packing regular and irregular pieces involves the bottom-left class of heuristics. These methods involve simply placing the input list of pieces into the bottom-leftmost location on the packing sheet [3]. In our implementation, we use these four placement heuristics that do not depend on the NFP:

1. **Bottom-Left (BLI).**- This is the best known heuristic of its type, and a modification to the bottom-left heuristic used for rectangular pieces presented in [138]. The piece starts at the top right corner of the object and it slides down and left with a sequence of movements until no other movement is possible (see Figure 2.5). If the final position does not overlap the object boundaries, the piece is placed in that position. The heuristic does not allow a piece to skip around another placed piece. The good performance of this heuristic greatly depends on the initial ordering of the pieces as reported by Dowsland et al. [45, 46]. Its advantage lies in its speed and simplicity.
2. **Constructive Approach (CA).**- This heuristic is based on the work presented by Hifi and M'Hallah [70]. The heuristic starts by placing the first piece at the bottom and left of the object. Then, the next piece is placed in one of the five positions:  $(\bar{x}, 0)$ ,  $(0, \bar{y})$ ,  $(\underline{x}, \bar{y})$ ,  $(\bar{x}, \underline{y})$  and  $(\bar{x}, y)$ , where  $\bar{x}$ ,  $\underline{x}$ ,  $\bar{y}$ , and  $\underline{y}$  are the maximum and minimum coordinates in  $x$  and  $y$  in relation to the first piece (see Figures 2.6 and 2.7). Given that some positions might coincide, each position appears only once in the list. For each position in the list, the next piece slides vertically and horizontally following down and left movements as shown in Figure 2.8. Positions with overlapping pieces or exceeding the object dimensions are discarded. All others are considered as candidate positions, and the one that places the piece deepest (bottom and left) is chosen. In the implementation by Hifi and M'Hallah [69], some special cases are discarded such as when a hole is formed. But, in our implementation of the heuristic, the four corners of the object were

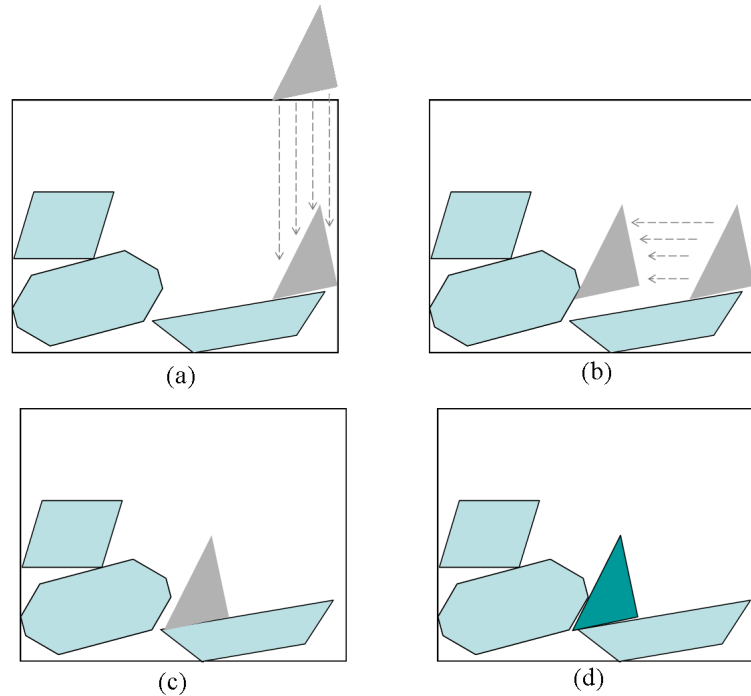


Figure 2.5: Bottom-left heuristic for irregular pieces.

also added as candidate positions. Using the corners as a departure point to slide the piece bottom and left, can make it possible to reach into certain gaps between pieces, gaps that would not be reachable if only the five initial positions are considered. The approach uses simple geometric operators, avoiding more sophisticated computations such as the convex hull.

3. **Constructive-Approach (Minimum Area) (CAA).**- This is a modification of the previous heuristic. The variation consists of selecting the best position from the list based on which one yields the bounding rectangle with minimum area, containing all pieces, and that fits in the bottom left corner of the object. The rectangle is shown in Figure 2.9

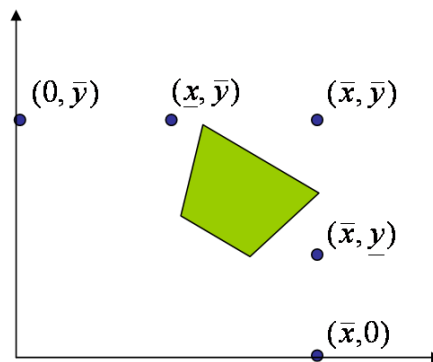


Figure 2.6: Positions to be considered in the Constructive Approach.

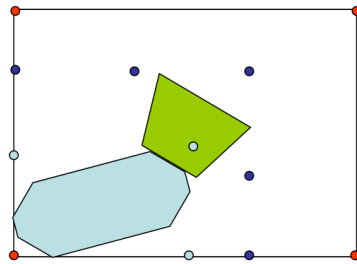


Figure 2.7: Positions to be considered in the Constructive Approach. An example with two pieces already placed.

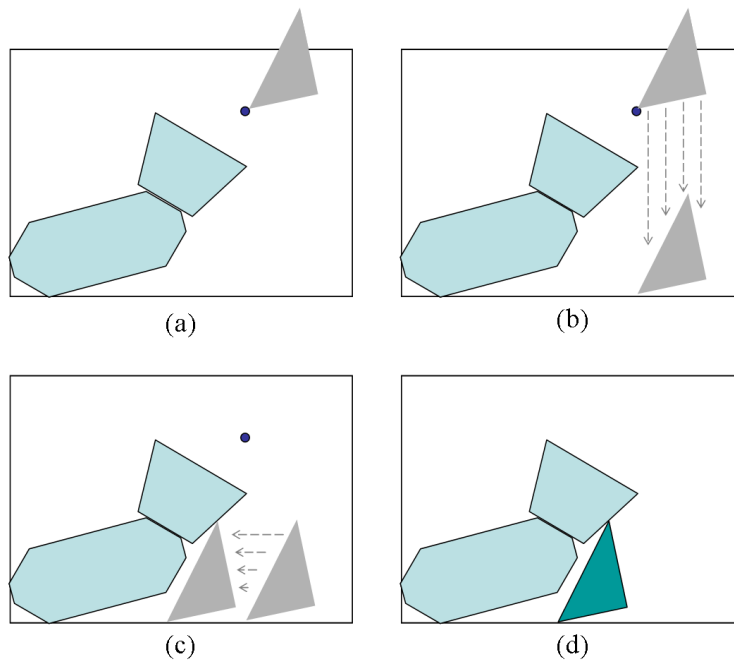


Figure 2.8: Constructive Approach Heuristic.

and its area is computed with the product of the maximum horizontal coordinate and the maximum vertical coordinate of all pieces already placed plus the new piece to be located in the proposed position. Figure 2.10 shows the rectangle with minimum area for two different proposed positions for a single piece. This criterion was chosen based on the idea of selecting a point with which all pieces, not only the last piece, are deepest (bottom and left).

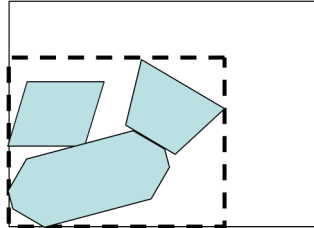


Figure 2.9: Rectangle with minimum area, located at the bottom-left corner and containing all pieces so far.

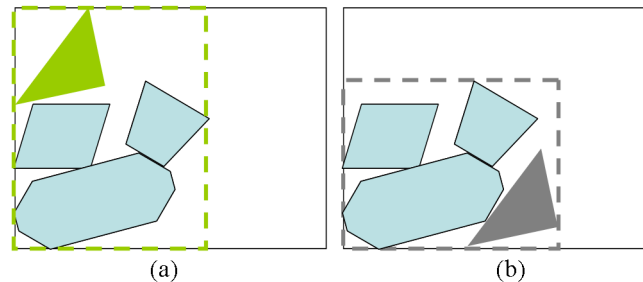


Figure 2.10: Candidate rectangles when locating a piece.

4. **Constructive-Approach (Maximum Adjacency) (CAD).**- The idea behind this heuristic is based on the approach suggested by Uday et al. [147]. However, when the first piece is to be placed, our implementation considers only the four corners of the object. For the subsequent pieces, the *possible points* are the same as those as in the constructive approach (CA, listed as our second placement heuristic), described above. Each position in the list is evaluated twice: first, the piece starts in that position and its adjacency (that is, the common boundary between its perimeter and the placed pieces and the object edges) is computed. Then, the piece is slid down and left and the adjacency is computed again. The position with the largest adjacency is selected as the position of the new piece.

These four placement heuristic implemented in this research involves sliding. The BLI heuristic slides all pieces from the top-right part of the object, while the remaining three heuristics start sliding from a set of different positions. Some placement positions that may not be reached by sliding but could be reached by *dropping* are not considered in this investigation. Positions like the one in the example of Figure 2.11 could only be achieved with

our constructive approach heuristics having the exact fitting position in the list of candidate positions described in CA heuristic above.

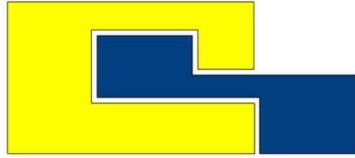


Figure 2.11: Two concave pieces that can not fit with a sliding operation.

### 2.6.3 Local Search Heuristics

Once a solution is found, further improvements can be found if a local search mechanism is applied to generate new input orderings [23]. The issue here is encountered in defining a neighborhood search strategy that can deal with the infinite solution space inherent in the irregular C&P problem [10].

Some of the local search heuristics applied to the 2D C&P problem are:

1. **Hill climbing.** This algorithm applies operators to the current solution to find a neighbor of increased quality. If an improved neighbor is found, it is adopted as the current solution and the search continues. If the neighbor is not an improvement on the current solution, it is discarded and the search continues with other neighbors. The best solution is returned at the end of the search [23]. A particular case of this kind of local search is called *2-exchange search procedure*, which is an improvement algorithm responsible for guiding the search through the solution space. This algorithm moves to a neighbor solution by exchanging a pair of pieces in the current sequence. Parameter  $\Delta$  controls the size of the neighborhood, allowing only exchanges between pieces within a distance of  $\Delta$  [62].
2. **Tabu search.** From a solution at hand, the process generates a given number of neighbors and moves to the best solution in this subset of the neighborhood. This best solution is then used to generate the next set of neighbors and the cycle continues. The use of a tabu list means that we will not revisit recently seen solutions within a given list length [23].

## 2.7 Meta-heuristic Search for the 2D C&P Problem

A meta-heuristic is a heuristic method for solving a very general class of computational problems by combining user-given black-box procedures -usually heuristics themselves- in a hopefully efficient way. The name combines the Greek prefix *meta* (*beyond*, here in the sense of *higher level*) and *heuristic*. Meta-heuristics are generally applied to problems for which there is no satisfactory problem-specific algorithm or heuristic; or when it is not practical to implement such a method. Most commonly used meta-heuristics are targeted to combinatorial optimization problems.

Meta-heuristic techniques are often very effective, however, there can be some reluctance to use them for money-critical problems. In practice, experience suggests that people often prefer to use very simple and readily understandable search methods even if those methods deliver relatively inferior results [126].

Many approaches for the C&P problem are proposed in the literature involving meta-heuristic search principles. Here are some of them related with the two-dimensional case, including some specific implementations. They are classified by kind of meta-heuristic technique utilized.

### 2.7.1 Evolutionary Computation (EC)

Evolutionary Computation includes a set of searching techniques typically used for optimizing combinatorial problems. Those techniques evolve a population through an iterative process. Finally, the best individual (and sometimes all of them) of the evolved population is considered the solution of the problem. The main evolutionary techniques include genetic algorithms (GA), genetic programming (GP) and learning classifier systems (LCS).

Among the main criticisms of bio-inspired algorithms in particular, and of stochastic based problem solving techniques in general, is the fact that they involve some randomness and unpredictability [127].

#### Genetic Algorithms

Genetic algorithms [59, 71] are meta-heuristic search procedures based on the mechanics of natural genetics and selection. They have been used for complex optimization problems with a large search space. The search is guided towards improvement applying the principle known as *survival of the fittest*. This is achieved by extracting the most desirable individuals from a generation of solutions and combining them to form the next generation. The quality of each solution is measured by a fitness function. It is intended that the higher the fitness value of an individual, the higher its reproduction probability. The motivation is to continue this process through a number of generations in order to reach convergence on optimal or near-optimal solutions.

Each point in the solution space is encoded in a string called chromosome. Each value of the chromosome is called a gen or bit, and the value it has is the allele. The classic approach uses binary coding where the parameters are represented by strings of 0's and 1's. Although representation with longer alphabets or real values have been implemented.

The genetic algorithm paradigm has attracted considerable attention as a promising approach for optimizing functions of continuous variables, but then there have been several applications to problems of a combinatorial nature. What is often found is that GAs have fairly poor performance for combinatorial problems if implemented in a naïve way, and most reported work has involved somewhat ad hoc adjustments to the basic method [122].

Since 1985, there have been GA implementations to the C&P context [74]. The GA manipulates the encoded solutions, which are then evaluated by a decoding algorithm transforming the packing sequence into the corresponding physical layout. Turton and Hopper [146] in 1997 did a review of the applications of GAs to packing problems. Genetic representation of the solution of the problem is vital to the performance of the genetic algorithm. One



of the most classical representations is the permutation [54, 81]. In several approaches, as in [103], the GA holds a population of solutions and each individual is assigned a fitness value, which indicates the quality of the solution the chromosome represents.

### **Naïve Evolution**

The basic idea behind naïve evolution (NE) is the same as for the genetic algorithm. However, no crossover operator is applied to manipulate the search space [137]. Only the mutation operator is used for the generation of the next population. A naïve evolution algorithm can be used to test the efficiency of the crossover operator in a genetic algorithm. Hopper and Turton [73] made a comparison among several meta-heuristic methods for the rectangular case. Their outcomes of the two evolutionary methods (GA and NE) are very similar with the NE algorithm performing slightly better for some problems (up to 2%).

### **Object-Based Evolutionary Algorithm (OBEA)**

Most of the evolutionary algorithms implemented so far for the nesting problem search in a one-dimensional space [121]. Kanchitpol and Dagli [121] proposed in 1998 a set of object-based mechanisms and object-based evolutionary operators to perform effectively on a two-dimensional space without restricting search alternatives. The representation of each individual is done through a collection of locations of the pieces which is also a solution of the problem. The implementation is conducted using grid representation where no overlapping is allowed. Some of the operators used are: *translation*, *rotation*, *rectangular-rotation*, *touch-point*, *piece-sliding*, *relocation*, *relocate-away*, *point-mutation*, *area-mutation*, *point-crossover* and *area-crossover*. The last two operators recombine two layouts. The translation operator, for example, is performed on a piece in a region by moving the piece one unit at a time towards preset gravitational forces. Details of all operators are explained in [121].

### **Genetic Programming (GP)**

Bounsaythip and Maouche [18] propose an evolutionary method to solve a garment shape nesting and placing problem. They use a hierarchical representation of the problem which is similar to representation used in genetic programming [116]. In their tree code, shapes are linked one to another by a layout operator (rotation, adjacency). Shapes can be viewed as *operands*. A layout of a total set of shapes is represented by a chromosome which contains trees as genes. Each tree represents a strip of layout, since the total set of shapes is partitioned into several strips. The crossover is the tree crossover now becoming classical in genetic programming. The mutation alters an operator in a tree or deletes a shape from a tree or permutes at random two single shapes within the same tree.

So far, not many researchers approach the irregular shape nesting problems with evolutionary algorithms. One main difficulty is to find an appropriate encoding of the problem [18].

### 2.7.2 Simulated Annealing

Simulated annealing (SA) [88] is a search method based on hill climbing in that better states are always accepted. However, worse states are also accepted, with some probability. This allows the algorithm to jump out of local minima so that more of the search space can be explored.

In 1999, Burke and Kendall presented a new method to pack convex polygons into bins by using simulated annealing and by utilizing the No Fit Polygon [29]. They showed that simulated annealing out performs hill climbing. The concept of a *polygon type* has been introduced. They implemented several neighborhood functions to explore the search space:

- **Collect:** Randomly selects a polygon and then scans through the remaining polygons and moves all polygons of the same type so that they are next to each other. The idea behind this function is that polygons of the same type will fit well together when packed.
- **Next Door:** Picks a polygon at random and swaps it with its next door neighbor. The motivation behind this function is to make small changes in the neighborhood in the hope that the search space will be systematically explored, leading to a good quality solution.
- **Random:** Selects two polygons at random and swaps them.

Later, in 2006, Gomes and Oliveira proposed an hybrid algorithm that includes Simulated Annealing and Linear Programming to solve the Irregular Strip Packing problem [63]. With these solution model, they got new best known results for all the benchmark instances used in the computational tests.

### 2.7.3 Tabu Search

Tabu Search (TS) appears to have been less popular (compared with GAs) in the solution of packing problems. This may be because of the infinite neighborhoods implied by a continuous stock sheet [10]. In Bennell and Dowland's implementation in 2001 [10] developed a neighborhood search and use compaction routines to improve local optima, ruling out simulated annealing as the underlying algorithm as it tends only to visit high-quality local optima toward the end of the search. Their approach used tabu thresholding, which searches the solution space by moving one piece to a different position in the stock sheet. Solutions containing overlap are permitted and penalized in the cost function.

### 2.7.4 Ant Colony Optimization (ACO)

ACO is a multi-agent meta-heuristic for combinatorial optimization and other problems. It is inspired by the capability of real ants to find the shortest path between their nest and a food source. The key to this ability lies in the fact that ants leave a pheromone trail behind while walking. Other ants can smell this pheromone, and follow it. When a colony of ants is presented with two possible paths, each ant initially chooses one randomly, resulting in 50% going over each path. It is clear, however, that the ants using the shortest path will be back

faster. So, immediately after their return there will be more pheromone on the shortest path, influencing other ants to follow this path. After some time, this results in the whole colony following the shortest path [47].

With ACO, agents (ants) solve difficult problems using single heuristics and communicating between them by signals left on the environment (*pheromones*) which store information of the solutions found and its quality [42]. ACO consists of:

1. Pheromones (for guiding ants through solution space).
2. Pheromone Update (to encode good solutions in the pheromones).
3. Pheromone Evaporation (to make bad solutions less probable).

In 1999, E. Burke and G. Kendall [28] implemented the ACO with regards to the nesting problem. Using the Traveling Salesman Problem (TSP) ant system as a model, an ant system has been developed for the nesting problem using the no fit polygon. Each polygon can be viewed as a city in the TSP and these are fully connected so that there is an edge between each polygon and every other.

### **2.7.5 Greedy Randomized Adaptive Search Procedure (GRASP)**

Gomes and Oliveira [61] proposed a GRASP (Greedy Randomized Adaptive Search Procedure) [114] approach to tackle the nesting problem. The idea behind this approach is to represent a solution by a sequence of pieces and perform the search over this representation, exchanging pairs of pieces in the sequence. A low level placement heuristic is needed for generating the cutting patterns (layouts), one for each particular sequence of pieces. This means that the search is conducted through a solution space of different sequences of pieces. The GRASP algorithm is an iterative process, where each iteration consists of a constructive phase and a local search procedure. The constructive phase is based on a greedy heuristic that builds a feasible solution, whose neighborhood is then explored by the local search procedure. At the end, the result is the best solution found over all of the iterations.

### **2.7.6 Agent-Based Approach**

Halavati et al. [67] present a population based approach for optimizing arrangement of irregular shapes. In this approach, each shape is coded as an agent and the agents' reproductions and grouping policies results in arrangements of the objects in positions with least wasted area between them. Agents try to make friendship relations with each other while the relation describes their relative positions. Once a group of friend agents is formed and the group has one agent from each shape, the friendships patterns describe the arrangement of their shapes. Based on the quality of the final arrangement, the friendship patterns are duplicated and some of the relations become permanent. Using this iterative approach, the quantity of the good groups increases gradually and more and more acceptable solutions appear.

Agent-based approaches are very uncommon for the 2D Cutting and Packing problem.

## 2.8 Hyper-heuristic Search

*Hyper-heuristics* are a recent development. The aim of research in hyper-heuristics is to discover new methods of solving difficult practical optimization problems, that are fast and are capable of delivering very good quality results for a wide range of problems.

Some of the reasons meta-heuristics (section 2.7), although effective, are not so popular in industrial domains include [126]:

- Meta-heuristics search techniques involve parameter choices that are critical for the technique performance. For many users those choices are not clear. Besides, such algorithms often involve probabilistic choices, that produce different solutions each run. This fact may make the procedure somewhat incompressible for some users.
- Development and implementation tend to be resource intensive, and each solution generation can be relatively slow.
- There is little knowledge or understanding of the average- or worst- case behavior of some of these techniques. So, there is not warranty of performance.
- Even if the technique generates a good-looking solution, it can be hard to understand how the solution was arrived at. It is often important to a group of people to feel that a proposed solution is intuitively acceptable.

That is, meta-heuristics are *problem-specific* solution methods, which require knowledge and experience about the problem domain and properties. They can be developed and deployed only by experts who have the sufficient knowledge and experience on the problem domain and the meta-heuristic search method [15].

Research on hyper-heuristics is an attempt to respond to such legitimate criticisms. That is why a hyper-heuristic model has to have **a clear and simple general form** [126]. The idea is to use evolutionary methods to discover a novel non-evolutionary *deterministic* algorithm, based on simple and familiar heuristics, that has good worst-case performance across a range of problems and is fast in execution [127]. However, Özcan et al. [112] talk about a *non-deterministic* strategy that schedules the next heuristic based on some probability distribution. More complicated and viable hyper-heuristics can be designed by making use of a learning mechanism that obtains a feedback from the previous choices to select the right heuristic at each step [112].

In combinatorial optimization, there is a trend to find more general solvers capable of solving many different instances efficiently; for example, Burke et al. [20] conducted an empirical study that ran the same high-level strategies (hyper-heuristics) with three different domains: 1D bin packing, permutation flow shop and personnel scheduling.

Hyper-heuristics methods do not search directly the solution space; they do it indirectly through the exploration of the space of heuristics and/or their combinations. This space is named the *associated space*. Experimental results reveal that the size of the associated space significantly affects the performance of the overall method [149]. Each member of the associated space is a combination of low level heuristics which represent a solution to the problem in hand. This indirect encoding has two main advantages:

- The associated space maps to solutions that are on average of a high quality. For example, a random sample of solutions in the associated space would be expected to have a superior quality than a random sample from the original solution space. This was tested by Vazquez et al. [149] using a production scheduling problem.
- The structure of the associated problem is, in general, simpler than that of the original problem, and therefore the searching task is easier.

Hyper-heuristics lie in between (1) machine learning and (2) optimization. Therefore, hyper-heuristics approaches can be classified according to (1) the source of the feedback and (2) the nature of the search space.

Three types of feedback can be distinguish: online learning (learning occurs while solving the problem) , offline learning (learning occurs with a training set of instances) and no learning.

Regarding the nature of the search space, two types of hyper-heuristics can be distinguished: heuristic selection (methodologies for choosing or selecting existing heuristics) and heuristic generation (methodologies for generating new heuristics from components of existing heuristics) [26]. Among the heuristic selection hyper-heuristics, there are the improving ones, also known as iterative or perturbation hyper-heuristics, and the constructive hyper-heuristics [149]. An iterative hyper-heuristic receives as an input a base solution and at every iteration a low level heuristic is applied in order to produce a hopefully better new solution; while a constructive hyper-heuristic is a high level heuristic that suggests a sequence of low level heuristics to be applied in succession in order to build a solution from scratch.

According to this classification of the hyper-heuristic approaches, the following definition can be stated: *A hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems* [26].

Each of the different heuristic search spaces can be combined with the different sources of feedback. Moreover, hybrid approaches are starting to emerge [26].

### 2.8.1 Heuristic Selection: Iterative Hyper-heuristics

An iterative hyper-heuristic receives as an input an initial *base* solution  $s^o$  and at every iteration a low level heuristic is applied in order to produce a new solution  $s'$ . If  $s'$  is better than  $s^o$  it becomes the new base for future iterations. If it is worse, it may either be discarded or still become the new base with a certain chance or some criteria. The mechanism to choose the low level heuristic to be applied next and the policy to accept or reject non-improving solutions are what differentiate most of the proposed iterative hyper-heuristics.

An iteration of a hyper-heuristic can be subdivided into two parts: heuristic selection and move acceptance. Cowling et al. [39] proposed three types of low level heuristic selection mechanisms to be used in hyper-heuristics; which are *Simple*, *Greedy* and *Choice Function*. There are three types of Single heuristic selection mechanisms. *Simple Random* mechanism chooses a low level heuristic at a time randomly. *Random Descent* mechanism chooses a low level heuristic randomly and applies it repeatedly as long as it produces improving results. *Random Permutation Descent* is similar to *Random Descent* except that first we choose a random permutation of the low-level heuristics  $N_1, N_2, \dots, N_\eta$ , and when application of a

low-level heuristic does not result in any improvement, we cycle round to the next heuristic in this permutation. *Greedy* method calls each low level heuristic at each iteration and chooses the one that produces the most improving solution. *Choice Function* is the most complex one. In this, a choice function  $F$  is introduced to decide on the choice of low-level heuristic to be called next. For each low-level heuristic the choice function  $F$  aims to measure how likely that low-level heuristic is to be effective, based upon the current state of knowledge of the region of the solution space currently under exploration. Cowling et al. [39] described four different methods for using the choice function. In the first (*Straight Choice*) method, they simply choose, at each iteration, the low-level heuristic which yields the best value of  $F$ . In the second (*Ranked Choice*) method they rank the low-level heuristics according to  $F$  and evaluate the changes in objective function value caused by a fixed proportion of the highest ranking heuristics, applying the heuristic which yields the best solution. The third (*Roulette Choice*) method assumes that for all low-level heuristics,  $F$  is always greater than zero. At each iteration a low-level heuristic  $N_i$  is chosen with probability which is proportional to  $F(N_i)/\sum_i F(N_i)$ . *Ranked Choice* and *Roulette Choice* are analogous to the rank-based selection and the roulette wheel selection from the genetic algorithms literature [59].

For each of these low level heuristic selection mechanisms two simple acceptance criteria are defined [39]. These are *AM*, where all moves are accepted and *OI* where only improving moves are accepted. For the *AM* versions of *Random Descent* and *Random Permutation Descent*, one move which makes the current solution worse will be carried out before moving on to a new neighborhood. The *AM* and *OI* versions of the *Greedy* approach are then identical to each other [39].

The hyper-heuristic will continue iterating until a stopping criterion is met, which can be a time limit [39].

## 2.8.2 Heuristic Selection: Constructive Hyper-heuristics

A constructive hyper-heuristic is a high level heuristic that suggests a sequence of low level heuristics to be applied in succession in order to build a solution from scratch. The task is, then, to find the *best* sequence.

Burke et al. [31] consider a hybrid framework in which the output of the constructive hyper-heuristic is used as input to the local search hyper-heuristic.

## 2.8.3 Heuristic Generation

This hyper-heuristic approach aims to generate new heuristics from a set of potential heuristic components. In the work by Burke et al. [25] this novel class of hyper-heuristics is discussed, in which genetic programming is the most widely used methodology. In this approach, human designed heuristics are broken down into their constituent parts, and a grammar is used to capture the structure of how the constituents relate to each other. The constituent parts, along with the grammar, are used to construct a search space. By the evolutionary process, new heuristics are formed and evaluated.

## 2.9 Hyper-heuristic Search for the 2D C&P Problem

Hyper-heuristic search in the two-dimensional C&P problems can be summarized in the following approaches. All of them focus on constructive hyper-heuristics. Three different visions of constructive hyper-heuristics can be found:

1. The hyper-heuristic as a high level heuristic that suggests a sequence of low level heuristics to be applied in succession in order to build a solution from scratch. Implementations for the regular 2D C&P are in [6, 42].
2. A variant of this concept of constructive hyper-heuristic is implemented in [139, 140, 143] where there exists a rule or a function to decide which heuristic applied given the current state of the problem. So, here the sequence of low-level heuristics is not determined in advance.
3. Another concept of constructive hyper-heuristic is proposed by Burke et al. [27], whose hyper-heuristic evolve a single heuristic for the instance at hand. They do not consider a sequence of single heuristics.

The hyper-heuristic could be developed through different meta-heuristic approach, as it is described in the following subsections.

### 2.9.1 Hyper-heuristics with Evolutionary Computation

Evolutionary techniques are the search algorithms most utilized for building hyper-heuristics for the 2D C&P problem.

### 2.9.2 Learning Classifier Systems

Learning classifier system (LCS) is an evolutionary technique combined with reinforcement learning and other heuristics to produce adaptive systems. The technique has been applied to a wide variety of domains. Terashima et al. [139] in 2005 present a method for combining concepts of hyper-heuristics and learning classifier systems for solving 2D regular (rectangular) Bin Packing Problems. In this work, the hyper-heuristic is formed using a XCS-type Learning Classifier System [35] which learns a solution procedure when solving individual problems. The XCS evolves a behavior model which determines the possible actions (selection and placement heuristics) for given states of the problem. When tested with a collection of different problems, the method finds very competitive results for most cases.

### 2.9.3 Genetic Programming

Recently, Hyde [78] and Burke et al. [27] employ genetic programming as a hyper-heuristic to search the space of heuristics that it is possible to construct from a set of building blocks. A heuristic in this hyper-heuristic system is a function that rates each possible allocation for each piece pending of been placed. The allocation with higher score indicates which piece will be placed next and where. The output of the genetic programming algorithm is the best

heuristic found which is applied to the solution space to produce a solution. Their model is for the two-dimensional strip packing problem, where the pieces are rectangles. This is called hyper-heuristic because it operates at a higher level of abstraction to previous meta-heuristic approaches, by operating on a space of heuristics and not directly on a space of solutions.

## 2.9.4 Genetic Algorithms

In 2005, Terashima et al. [140] developed a hyper-heuristic solution model using a GA with fixed-length chromosome that has several groups with four parameters each. They proved their model with the 2D regular BPP finding very competitive results for most cases, when tested with a collection of different problems. The testbed is composed of problems used in other similar studies in the literature.

Later, a hyper-heuristic model is build using a non-traditional GA, with variable length chromosomes. Because of this, the GA employed seems like a messy-GA [43, 60]. This model was tested in regular (rectangular) [138] and irregular [143] pieces. Up to date, this has been the only hyper-heuristic model applied in irregular 2D BPP. In Chapter 5 a more detailed description is given.

## 2.9.5 Hyper-heuristics with Ant Colony Optimization

The first attempt to combine hyper-heuristics with an ACO algorithm is done by Cuesta-Cañada et al. [42] in 2005. The resulting algorithm was applied to the two-dimensional Bin Packing Problem, and encouraging results were obtained when solving classic instances taken from the literature. In this work, they use the term *single heuristic* to denote the combination of five variables: Quantity, Rotation, Item Order, Bin Selection Heuristic and Placement Heuristic. Each of these five variables can manage several values. The term *hyper-heuristic* describes the sequential combination of at most five *single heuristics*. That is, a hyper-heuristic is represented with a twenty-five cells array (five cells for each one of the five heuristics). They implemented 25 pheromone matrices, each one representing the transition between two consecutive variables. Each matrix had a number of rows equal to the number of possible values for that variable, and a number of columns equal to the number of possible values of the next variable. A entry in row  $i$  and column  $j$  encodes: provided value  $i$  of current variable is selected, select value  $j$  of next variable with probability  $Ph_v(i, j)$ . A path is a route from a point in the first matrix, going to a point of every matrix from then. A single group of ants working simultaneously its entire paths is called an *iteration*. When an iteration finishes, the quality of each path is updated, updating this way the probabilities for each possible path.

## 2.9.6 Hyper-heuristics with Hill Climbing

The hyper-heuristic developed by Araya et al. [6] in 2008 manages a sequence of greedy low-level heuristics, each element of the sequence placing a given number of objects. Following the sequence of low-level heuristics construct the solution for a given instance. A hill-climbing algorithm is performed on this sequence by testing a different move (adding, removing, replacing a low-level heuristic) in each iteration. If the new sequence is better, it



replaces the current one. In order to escape local minima, they perform several restarts. Their model is applied for the rectangular two-dimensional strip packing problem.

## 2.10 Hyper-heuristic Search for other Optimization Problems

The following are NP-hard combinatorial problems as well as the C&P problem. It can be observed that the approaches applied to these problems have similarities to those applied to the C&P problem.

### 2.10.1 Timetabling Problems

Timetabling problems are real world constraint optimization problems. These problems have been intensively studied with the hyper-heuristic approach. Timetabling problems require assignment of time slots (periods) and possibly some other resources to a set of events, subject to a set of constraints. Numerous researchers deal with different types of timetabling problems based on different types of constraints utilizing variety of approaches. *Employee timetabling*, *course timetabling* and *examination timetabling* are the research fields that attract the most attention [15]. *Personnel scheduling*, *rostering*, *labor scheduling* are other terms to describe the same or similar problems.

One of the first hyper-heuristic approaches for this problem was done by Terashima et al. [142], although their combination of choices of heuristics was not called hyper-heuristic yet. They build the timetable sequentially, dividing the task into two phases. In the first phase, a certain strategy is chosen, and also a heuristic for choosing which exam to consider next, and a heuristic for deciding which time slot to put it in. After a certain condition is met, the second phase starts; this also applies a certain strategy and a pair of heuristics. They used a GA which chromosome encodes: (a) choices of strategy and heuristics in phase 1; (b) when to switch: either after placing  $\alpha$  exams, or after placing the largest N% of exams; (c) choices of strategy and heuristics used in phase 2.

Bilgin et al. [15], for example, developed an iterative hyper-heuristic that chooses a heuristic to apply to a candidate solution of the problem at hand, at each step. In their paper, seven heuristic selection methods and five different acceptance criteria are analyzed empirically. The thirty-five hyper-heuristics generated by coupling all heuristic selection methods and all acceptance criteria with each other, are evaluated on a set of twenty-one exam timetabling benchmark problem instances. Their experimental results denote that no combination of heuristic selection and move acceptance strategy can dominate over the others on all of the benchmark functions used.

Burke et al. [33] developed a case-based heuristic selection approach that collects and reuses previous experience of the heuristics that were employed successfully within particular situations onto current *similar* situations in timetabling. Although authors do not called this a hyper-heuristic approach, their aim is to develop intelligent systems that can *guess* at which heuristic will work well on which problem, and thus is capable of dealing with any problem efficiently. This is, indeed, very similar to the hyper-heuristic idea. They also developed a knowledge discovery process for finding out the relevant problem features and their weights.

Their motivation behind Case Based Reasoning is that humans often solve new problems by reemploying knowledge that has been collected from previous experience.

### 2.10.2 Constraint Satisfaction Problems (CSP)

Terashima et al. [141] present a GA-based method that produces general hyper-heuristics for the dynamic variable ordering within Constraint Satisfaction Problems. The GA uses a variable-length representation, which evolves combinations of condition-action rules producing hyper-heuristics after going through a learning process which includes training and testing phases. Such hyper-heuristics, when tested with a large set of benchmark problems, produce encouraging results for most of the cases. This is actually the same hyper-heuristic construction model than the one explained in detail in Chapter 3 applied for the CSP.

## 2.11 Meta-learning

Within the field of machine learning, the term *meta-learning* has been associated with the idea of, given a new dataset, automatically selecting the best learning algorithm for the problem at hand. Meta-learning ideas have traditionally been applied to learning algorithms to solve classification problems, where the goal is to relate performance of algorithms to characteristics or measures of classification datasets.

This task is also known as the Algorithm Selection Problem, which was first described by J. R. Rice in 1976 [123]. The problem is formally defined as: learning a mapping from feature space to algorithm performance space, and acknowledged the importance of selecting the right features to characterize the hardness of problem instances [132]. This definition includes three main elements:

1. *Problem space*. All possible instances of the problem. The problem can be characterized by a large number of independent features, that may be relevant for the algorithm selection and performance.
2. *Algorithm Space*. The algorithm or heuristic repository.
3. *Performance Measure*. The criteria used to evaluate the performance of a given algorithm with a problem instance.

From the definition of the Algorithm Selection Problem, there are three main aspects that must be considered in order to find an adequate way of solving problem instances based on a set of characteristics: (1) The selection of the set features, (2) The selection of the set of algorithms that together allow to solve the largest number of instances, and (3) The selection of an efficient mapping mechanism that permits to select the best algorithm for each instance [41].

Smith-Miles presented a framework for the generalization of algorithm selection and meta-learning ideas to algorithms focused on other tasks such as sorting, forecasting, constraint satisfaction and optimization [133]. There have been surprisingly few attempts to generalize the relevant meta-learning ideas to optimization, although several approaches can be

found in the related area of constraint satisfaction [94]. In the work by Smith-Miles [134], meta-learning ideas are used for modeling the relationship between instance characteristics and algorithm performance for the quadratic assignment problem. The study considered a set of 28 problem instances and three meta-heuristic algorithms. Both unsupervised and supervised neural network models were used to learn the relationships in the meta-dataset and automate the algorithm selection process. The unsupervised model, self-organization maps [90], was used to select the best algorithm by creating visual explorations of the performance of different algorithms under various conditions describing the complexity of the problem instances. Given the limited size of the data, this is a preliminary study, but it demonstrates the relevance of meta-learning ideas to the optimization domain. In a later study, Smith-Miles [135] found correlations between problem features and the effectiveness of scheduling heuristics using a large collection of instances in a production scheduling problem.

Other authors have followed up this type of study in optimization. For example, Kanda et al. [85] described each instance of the traveling salesperson problem by meta-features that capture characteristics of the problem that affect the performance of the optimization algorithms.

## 2.12 Summary

In this chapter, the 2D irregular BPP has been defined as a NP-hard problem and as a particular case of the C&P problem named 2-dimensional irregular Single Bin Size Bin Packing Problem (2D irregular SBSBPP) according to the typology of Wäscher et al. [154]. This is an attractive research topic due to its complexity and practical implications. Main solution approaches to NP-hard combinatorial problems are reviewed, making emphasis in those applied to the 2-dimensional C&P problem. The 1D BPP is described as well, since it is also tackled in this dissertation. Specifically a recent approach, called hyper-heuristics, is described in detail as an algorithm that operates at a higher level of abstraction compared with the previous state of art. Research done in hyper-heuristics for the BPP and other optimization problems is reviewed. Finally, a review of some meta-learning ideas is performed focusing in those works that have attempted to explain algorithm performance based on problem attributes. This last section describes the theoretic background for the research presented in Chapter 9.

In summary, this chapter presents all the related background relevant to the contribution of the present dissertation. The next chapter presents the details of the evolutionary framework proposed for solving the Bin Packing Problem.



# Chapter 3

## Hyper-heuristic Solution Model

This chapter presents in detail the solution model based on hyper-heuristic construction. As we will see, this hyper-heuristic generator algorithm is not problem dependent, since it has been adapted to several variants of the C&P problem: one-dimensional [128], two-dimensional regular [138] and two-dimensional irregular (convex) [143]; as well as to other optimization problem: the Constraint Satisfaction Problem (CSP) [141]. Our approach differs from meta-heuristics and other approaches in that instead of controlling simpler heuristics for one or a narrow set of problems, a hyper-heuristic is a re-usable method that chooses between a wider range of heuristic approaches to robustly tackle a wide range of problems.

The fully explanation of this solver is necessary because it is taken as a basis for this research.

Terashima et al. [138] and Fariás [52] present a genetic algorithm based method that produces general hyper-heuristics that solve two-dimensional cutting and packing problems. They support their idea in the work by Ross et al. [128]. The GA uses a variable-length representation, which evolves combinations of condition-action rules producing hyper-heuristics after going through a learning process which includes training and testing phases. Such hyper-heuristics, when tested with a large set of benchmark problems, produce outstanding results (optimal and near-optimal) for most of the cases. The testbed is composed of problems used in other similar studies in the literature. Some additional instances of the testbed were randomly generated.

Then, we applied the same solution model for the convex polygon version of the 2D BPP, obtaining encouraging results [98, 143]. As far as we know, this is the first hyper-heuristic model applied to irregular pieces of the 2D BPP. This research is described in Chapter 5. This solution model was later implemented with a wider range of instances, a new representation scheme and a revised heuristic repository in Chapter 8.

The main reasons for which this solution model was chosen as the building block for this dissertation, are the promising results that this evolutionary framework produced when applied with some variations of the BPP [128, 138, 143] and the observation that some aspects can still be improved.

According to the classification of hyper-heuristic approaches suggested by Burke et al. [26], our developed solution model falls into the category of *heuristic selection* because the hyper-heuristics produced *select* the best single heuristic to be applied (rather than generate a new heuristic). Also, we are dealing with an *offline learning hyper-heuristic*: the idea is to

gather knowledge in the form of rules or programs, from a set of training instances, that would hopefully generalize to the process of solving unseen instances [26].

### 3.1 General Process for Generating Hyper-heuristics

The hyper-heuristic, defined as a set of rules, is the result of an evolutionary process using a variable-length chromosome GA. This hyper-heuristic framework operates at a higher level of abstraction since it has no knowledge of the problem domain. On one hand, it *communicates* with the instances states of the specific problem through simplified numerical representations. On the other hand, it only has access to a set of low-level heuristics that it can call upon, but with no knowledge as to the purpose or function of a given low-level heuristic (see Figure 1.4 from Chapter 1). The GA search space consists of all possible set of matchings between problem state representations with subsets of low-level heuristics.

The general process of our solution model starts evaluating individual heuristics, each single heuristic is applied to all problem instances, and the best heuristic per instance is recorded for later comparison. The available problem instances are divided into a training and a testing set. Then the GA is used with the training set only, until a termination criterion is met and a general hyper-heuristic has been evolved. All instances in both the testing and training sets are then solved with this general hyper-heuristic and results are compared with those obtained by single heuristics. The complete process is shown in Figure 3.1.

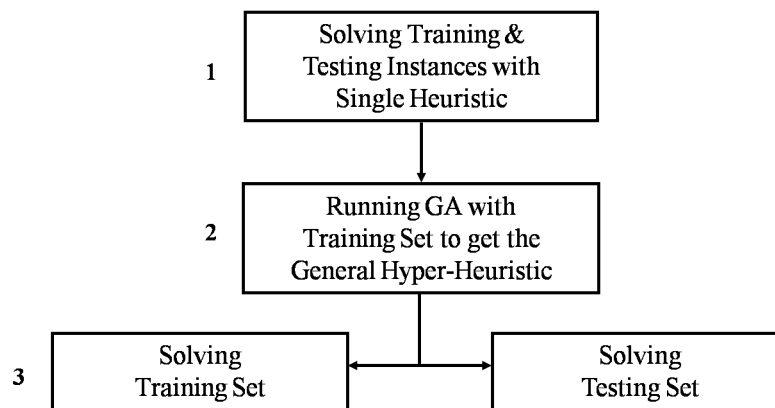


Figure 3.1: Solution Model for Hyper-heuristic generation through GA.

Hyper-heuristics are sometimes described as *heuristics that search a space of heuristics*. In this research, each chromosome in the evolutionary process is a recipe for how to apply a number of heuristics in order to construct a solution. The evolutionary search process itself searches a space of particular ways of combining heuristics. Thus, the reader might regard either the chromosomes themselves, or the whole search process itself, as being hyper-heuristics. In this research we choose to refer to the chromosomes as being hyper-heuristics, and trust that this will not cause undue confusion.

The key idea in our constructive approach is to build a complete solution by deciding what to do at each stage, including the initial stage. Here, *what to do* means using some chosen heuristic to place a piece and thus extend the solution. Each *stage* is described by

some kind of simplified representation of the problem state. We hypothesize that if two states are very similar then we would want to do the same thing in either state. The hyper-heuristic have the following general form.

UNTIL the solution is complete:

1. calculate the vector  $P$  that describes the current problem state;
2. find the nearest representative vector  $R$  to  $P$ ;
3. execute the heuristic associated with  $R$ , thus changing the current state.

In order to implement this, we need several ingredients:

- we need to choose a vector-based way of representing the problem state;
- we need to choose a set of heuristics than can be used;
- we do not know how many representative vectors we need or what their values should be or which heuristics to associate with each of them. So we need a search method that can do this for us.

These ingredients are described below.

## 3.2 Representation of Problem Instances

Each instance to be solved by the hyper-heuristic is characterized by a numerical vector that summarizes some of its relevant features (see Figure 1.3 from Chapter 1). Each numerical term of the vector quantifies an aspect of the instance at hand. For example, for the 2D irregular BPP, the average size of the pieces to be placed could be an important feature. According to this numerical vector, the hyper-heuristic decides which single heuristic to apply every time. The numerical representation is applied for *complete instances* to be solved (where no piece has been placed yet) as well as for *instances partially solved* (where some pieces have already been placed). That is, for a given problem instance, numerical representation is computed in every intermediate state until it is completely solved. An instance *state* is every intermediate phase in the solution process until all pieces are placed and a solution for the instance is found.

## 3.3 Representation of Chromosomes in the GA

We employ a GA with variable-length individuals. Each chromosome is composed of a series of *blocks*. Each block includes several numbers. All numbers in a block, except the last one, represent an instance state which is the numerical vector mentioned above (Figure 3.2). The label is the last number, which identifies a single heuristic. A chromosome consists of a number of points in a simplified state space, each point being labelled with a single heuristic (Figure 3.3). An individual solves a problem instance as follows: given an instance and having computed its numerical representation ( $P$ ), find the closest block in the chromosome (with Euclidean distance) and apply the single heuristic recorded on the label. This will place

one or several items or pieces and will produce a new problem-state representation ( $P'$ ). The process is repeated until all pieces are placed and a complete solution has been constructed. A chromosome therefore represents a complete recipe for solving a problem, using this simple algorithm: until the problem is solved, (a) determine the current problem state  $P$ , (b) find the nearest point to it, (c) apply the heuristic attached to the point, and (d) update the state. The GA's task is to find a chromosome that is capable of obtaining good solutions for a wide variety of problems; the chromosome is the hyper-heuristic that we seek.

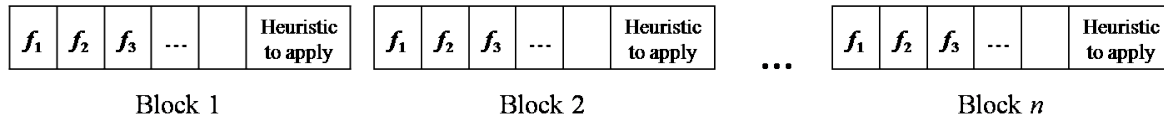


Figure 3.2: A chromosome is a set of blocks.

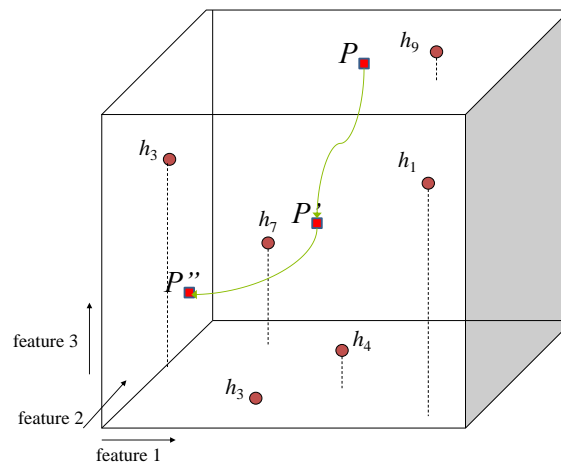


Figure 3.3: A chromosome is a set of blocks. Each block represents a point in the hypercube (space of states) labelled with a single heuristic. The solution process of a problem using a hyper-heuristic consists of finding the closest single heuristic at every solution stage.

Figure 3.3 shows all possible three-feature representations as a cube in a three dimensional space. Every dimension  $x, y$  and  $z$  represents a different problem feature. With a given set of instances, not every feature combination may be possible; so, not every point in the feature space may represent a valid state in a particular implementation. For example, it is not very likely that the features *percentage of small items* and *average of items sizes* have large values at the same time. A hyper-heuristic links several points from the representation space with specific heuristics from the heuristic repository  $H$ . Note that not every heuristic in  $H$  is in the hyper-heuristic, but a single heuristic could be connected with more than one point in the representation space. Also, the model permits the points defined in the hyper-heuristic to lie outside but close to the representation space. This means that if the problem state is on one of the cube's faces, the nearest heuristic could be outside of the cube. Since the hyper-heuristic is evolved using a variable-length GA, the number of labelled points at the representation space is not determined in advance.



### 3.4 Fitness Function

The quality of a solution, produced by any pair of selection and placement heuristics for a given instance, is based on the percentage of usage for each object, which is given by:

$$U = \frac{\sum_{j=1}^n A_j}{A_O} \quad (3.1)$$

where  $A_j$  is the area of piece  $j$ ,  $n$  is the number of pieces inside the object and  $A_O$  is the total object area. Once the fractional utilization is computed for each object, the quality of the solution is given by:

$$Q = \frac{\sum_{i=1}^{N_o} U_i^2}{N_o} . \quad (3.2)$$

where  $N_o$  is the total number of objects used and  $U_i$  is the fractional utilization for each object  $i$ . Note that each  $U_i \leq 1$ . This measure of fitness rewards objects that are filled completely or nearly so, and avoids the problem of too many ties among different heuristics that occur when quality is simply the number of objects used.

It is necessary to compute the fitness produced by each individual combination of selection and placement heuristics, for each instance. The best heuristic combination and its result, for each specified instance  $i$  are stored (let us call it  $BSH_i$ ). These results are prepared before running the GA.

Now, each chromosome solves some instances and its fitness is computed as the average difference between the solution quality obtained by the chromosome with respect to the result given by the best single heuristic for every particular instance. The fitness is an average given by:

$$f = \frac{\sum_{k=1}^m (Q_k - BSH_k)}{m} \quad (3.3)$$

where  $BSH_k$  is the best quality solution obtained by a single heuristic for the  $k$ -th assigned instance,  $Q_k$  is the quality solution obtained by the hyper-heuristic for the  $k$ -th assigned instance and  $m$  is the number of instances solved so far.  $BSH_k$  and  $Q_k$  are computed using Equation 3.2.

After each generation  $l$ , a new problem is assigned to each individual in the population and its fitness is recomputed as follows:

$$f_l = \frac{f_{l-1} \cdot m + f_{new}}{m + 1} \quad (3.4)$$

where  $f_{l-1}$  is the fitness for individual in the previous generation;  $m$  is the number of problems this individual has seen so far;  $f_{new}$  is the fitness obtained by individual for the new problem and computed with the fitness function given by equation 3.3. Note that  $f$  could be negative if the hyper-heuristic has poor performance. Then, we expect that the evolutionary process removes this poor-quality chromosome.

## 3.5 The GA Cycle

The steps of the GA cycle are:

1. Generate initial population. Each individual is comprised of a series of blocks from 10 to 15 inclusive. The number of blocks is chosen according to a discrete uniform distribution. The number of elements of each block depends on the number of features chosen for representing each problem instance state. The elements of each block, except the last one, are real random numbers between  $-2$  and  $3$ , with a Gaussian distribution with mean  $0.5$  and standard deviation  $0.5$  and truncated accordingly. The last number of each block is an integer from  $0$  to  $|H| - 1$ ; where  $|H|$  is the number of the available single heuristics.
2. Randomly assign 5 problems from the training set to each chromosome and compute the chromosome's fitness based on these problems (with Equation 3.3).
3. Apply selection, crossover and mutation operators to produce two offspring. We select two different individuals at each cycle, each individual selected is the best from a randomly chosen pair of chromosomes (tournament size 2). In the model proposed by Ross [128], one of the individuals was chosen by tournament of size 2, while the other was selected randomly. In this research, we increased the selective pressure in which both individuals come from a tournament. Genetic operators are described below. After crossover and mutation, the number of blocks in any chromosome may vary.
4. Randomly assign 5 problems to each new individual and obtain its fitness (with Equation 3.3).
5. Replace the two worst individuals with the new offspring provided they are of better fitness.
6. Assign a new problem to every individual in the new population and update fitness (with Equation 3.4).
7. Repeat from step 3 until a termination criterion is reached. In our implementation, the termination criterion is a reached number of generations.

Figure 3.4 shows the genetic algorithm general process for producing hyper-heuristics.

### 3.5.1 Genetic Operators

In this investigation we used two crossover and three mutation operators. These operators were taken from the previous implementation of the solution model for the 1D BPP [128] and the 2D regular BPP [52, 138]. The probability for applying each type of crossover or mutation operator was suggested by some early testing in the investigation for the 1D case [128].

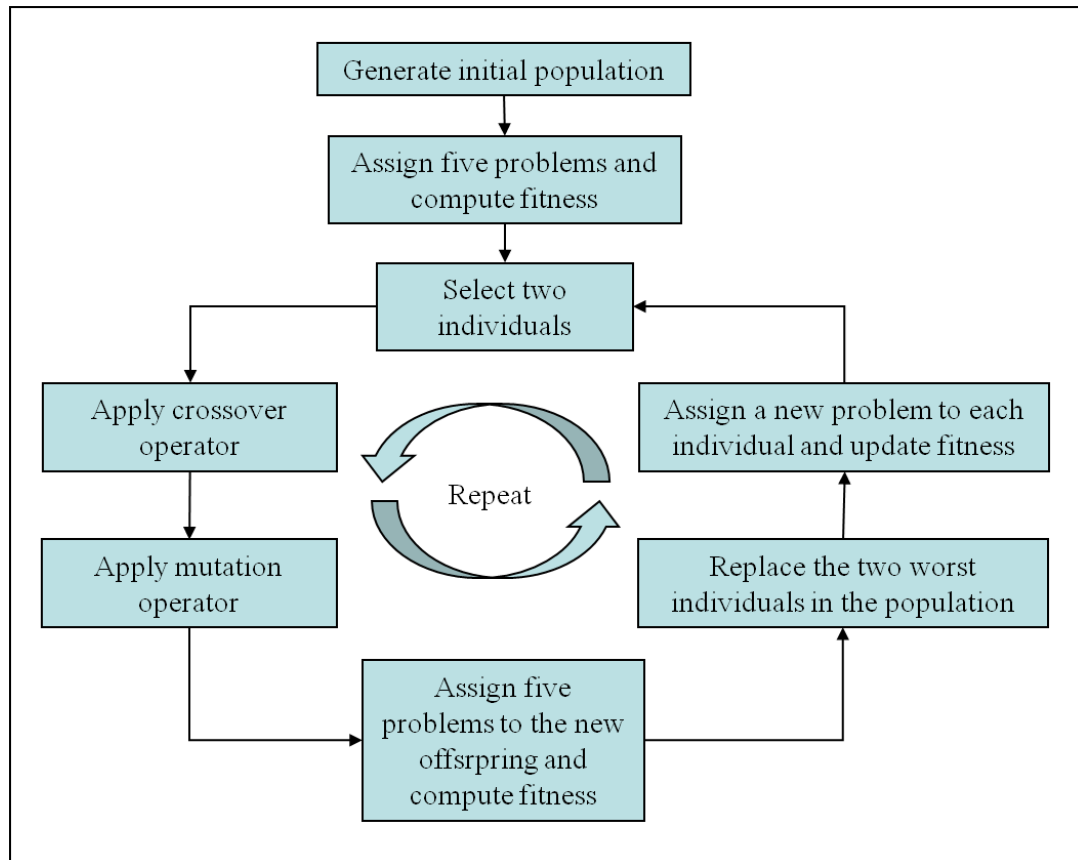


Figure 3.4: General process of the genetic algorithm.

### Crossover Operators

Both of the following crossover operators employed have the same probability of being chosen.

1. **One-point crossover.** This operator works at block level, and it is very similar to the normal one-point crossover. This operator exchanges 10% of blocks between parents, meaning that the first child obtains 90% of information from the first parent, and 10% from the second one, and vice-versa. This operator shuffles blocks. The blocks passed from a parent to an offspring are not in consecutive order. The 90% of the number of blocks is computed for each individual and the result is truncated to obtain an integer. This is the reason why the number of blocks in the two offspring may vary slightly.

This operator is somewhat different comparing with the version implemented for the 1D case [128], where each block of the first parent has a 90% chance of being passed to the first child and a 10% of being passed to the second child and vice-versa. If the decisions about which offspring will keep block are independent, then the number of blocks in the new individuals varies.

2. **Two-point crossover.** This operator is very similar to the normal two-point crossover.

For each individual, we first select two blocks and then a point inside each block is chosen. Since the number of blocks in each chromosome is variable, the cut points in each parent are chosen independently. However the points selected inside each corresponding block are forced to be the same for both parents, to avoid changing the meaning of any numbers; so that the recombination produces an exact number of blocks. The blocks and points are chosen using a uniform distribution. The first block selected in each chromosome is always in the first half of the individual length and the second selected block is any block after the first one. The number of blocks in the two offspring may be different from the number of blocks of their parents and it is not adjusted.

Figure 3.5 shows an example of this procedure. Parent 1 has four blocks and parent 2 has three blocks. Blocks 2 and 4 are selected from the first parent while blocks 1 and 2 are selected from the second parent. Internal points  $b$  and  $c$  are chosen. Note that this internal points in the blocks are the same for the first and second blocks respectively for both parents.

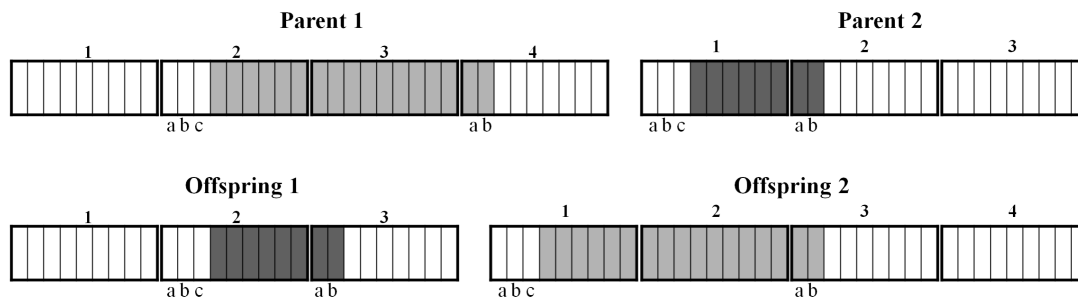


Figure 3.5: An example of the two-point crossover operator.

There is an exception for this type of crossover operator. If one parent has a length of up to two blocks, then the individual is completely removed and recreated randomly with a number of blocks from 10 to 15 inclusive. The other selected individual for crossover is copied exactly. The idea is to penalize chromosomes with a very small number of blocks.

### Mutation Operators

After the GA has decided to mutate, these three mutation operators have probabilities of 0.25, 0.25 and 0.50 respectively. The first two operators are most disruptive, and they are jointly chosen with the same probability than the third.

1. **Add-block mutation.** Randomly generates a new block and adds it at the end of the chromosome. As an exception, this operator removes a block instead of adding one when the individual length is 20 blocks (or more). The idea is to keep the number of rules (blocks) in each chromosome as a manageable number. In early experimentations, we observed that the number of rules employed to solve a given instance tends to be small: fewer than 10 rules in most of the cases. Tables 8.9 and 8.10 show the number of changes of single heuristics when solving instances. Therefore, these tables serve as a

reference that we do not need too many blocks in a chromosome to have a good-quality hyper-heuristic, at least when dealing with our instance testbed.

2. **Remove-block mutation.** Randomly selects and eliminates a block within the chromosome. An exception occurs when the chromosome length is less than 6 blocks. In this case, the operator adds a block instead of removing one.
3. **Normal mutation.** Randomly selects a position inside a random block. If the selected position is the last one, then replace that value with a random integer selected from 0 to  $|H| - 1$ ; where  $|H|$  is the number of the available single heuristics. Otherwise, replace the selected value with a new number between  $-2$  and  $3$ , generated with a normal distribution with mean  $0.5$  and standard deviation  $0.5$  and truncated accordingly.

## 3.6 Rotation Scheme

The rotation scheme is another issue to be analyzed. In the first implementation of the model (described in Chapter 5 and [143]), we ran preliminary studies to determine a rotation scheme used in the investigation of convex instances. The first one rotates each piece by multiples of 90 degrees, that is, 0, 90, 180 and 270. The second approach rotates each piece in multiples of 5 degrees. Interestingly, this second approach showed a very marginal improvement compared to the first approach, and had the disadvantage of the extra computational cost. This could be due to the fact that pieces are presented to the solver in the same orientation that corresponds to the optimal solution and non-orthogonal rotation does not help. Besides, another disadvantage of the second approach is that sometimes precision was lost because of rounding the coordinates after rotation. That is why the first rotation approach is used throughout current research.

## 3.7 Codification of Problem Instances

Each problem instance is represented in the hyper-heuristic framework as a text file. The first line is the number of pieces while the second line shows the object dimensions. From the third line and on, each piece is represented by its vertices coordinates which are ordered counterclockwise (Figure 3.6). Coordinates origin is the bottom-left corner of an orthogonal rectangle enclosing the piece (Figure 3.7).

## 3.8 Summary

In this chapter, the hyper-heuristic solution model based on an evolutionary framework has been presented in detail since it is the basis of the dissertation. The next chapter explains the research methodology employed in this investigation as well as the origin of our testbed instances and some geometric functions developed for solving irregular instances.

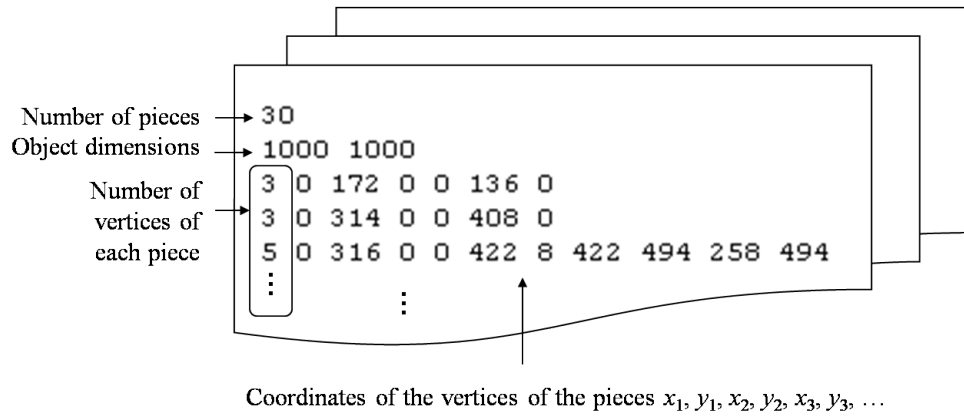


Figure 3.6: Representation of problem instances in a text file.

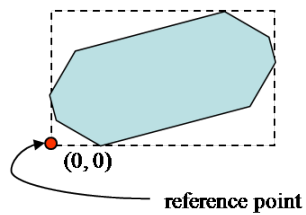


Figure 3.7: Origin of the coordinates of the pieces of a problem instance.

# Chapter 4

## Research Methodology and Experimental Setup

Previous chapters developed the related background (Chapter 2) and described in detail the hyper-heuristic solution model which is the building block of this dissertation (Chapter 3). At this point, the state of the art has been presented completely. Now, this chapter develops the main steps followed during the research process. Working problem instances and algorithms implemented for geometric computations are presented as well, since they are employed in the several experiments carried out in the different research stages. The following five chapters (5, 7, 6, 8 and 9) develop the own scientific contribution to the field.

### 4.1 Methodology

The steps followed in order to fulfill the objectives are listed below. These activities are organized in a timely manner approximately.

1. Background and state of the art. A research was conducted about the problem to be tackled and related issues. This literature review was done extensively at the beginning of the doctoral studies and updated periodically to keep the background up to date during the dissertation period.
  - (a) Research about the C&P problem, its variants, relevance and solution approaches up to date. We focus then in the 1D and 2D BPP.
  - (b) Investigate about the geometry and placement heuristics involved when allocating concave polygons.
  - (c) Research about search approaches in general and hyper-heuristics approaches in particular.

After this startup, research questions and objectives could be established in order to contribute to some progress to the research area.

2. Perform a first implementation of the solution model to the 2D irregular BPP (convex polygons).

- (a) Implement some placement heuristics for irregular pieces.
  - (b) Program and implement geometric functions for dealing with irregular convex polygons.
  - (c) Design and program an algorithm to generate instances with convex polygons in order to generate own instances with known optimum.
  - (d) Gather a set of instances for experimentation purposes. This was done generating some new instances.
  - (e) Develop the instance representation scheme, choosing some features regarding irregular shapes.
  - (f) Design and perform several types of experiments, to evaluate the quality of the hyper-heuristics generated.
  - (g) Analyze the results and draw conclusions.
  - (h) Write an article with the findings and conclusions [143].
3. Research about instance representation. This is a detailed research on one critical aspect of the solution model developed. It focuses on the characteristics of the instances.
- (a) Make a deeper literature review about the instances representation schemes.
  - (b) Develop a methodology for finding out what are the most relevant features of a given problem.
  - (c) Apply the methodology in the 2D irregular BPP, with instances including convex polygons.
  - (d) Evaluate the representation found for the 2D irregular BPP, replicating the experiments from the first implementation of the model ([143] and Chapter 5).
  - (e) Compare the results with those from the first implementation of the model.
  - (f) Write an article with the findings and conclusions [99, 101].
4. Adapt the DJD heuristic to the 2D Bin Packing Problem. We achieved an effective implementation of one of the selection heuristics considered.
- (a) Make a deeper literature research about the DJD heuristic. Here we found that several papers [104, 105, 144] refer the article “Philipp A. Djang and Paul R. Finch. Solving One Dimensional Bin Packing Problems. *Journal of Heuristics*, 1998” as the first source presenting the DJD heuristic. Nevertheless, it seems that this article does not belong to that journal. Kos and Duhovnik [91] cited the article as *submitted to the Journal of Heuristics*. Other publications [128, 129] does not mention any journal, and some others [22, 126] refer to a link which is not active any more. The point is that, despite our efforts, we could not find the original source of the DJD heuristic.
  - (b) Design and program functions for keeping track of the pieces already revised when applying of the DJD heuristic.



- (c) Make preliminary experiments for choosing a good value for two parameters of the DJD heuristic: initial fullness of the objects and incremental of allowed waste.
  - (d) Solve a set of test instances with several selection heuristics including some variants of the DJD.
  - (e) Analyze the results and draw conclusions.
  - (f) Write an article with the findings and conclusions.
5. Construct a hyper-heuristic model for several kinds of the BPP, since 1D to the 2D irregular BPP (non-convex). The model developed previously [143] will expand its scope to be more general.
  - (a) Establish the repository of low-level heuristics adequate to handle concave polygons.
  - (b) Program and implement geometric functions for dealing with concavities.
  - (c) Design and program an algorithm to generate instances that include concave polygons in order to generate own instances with known optimum.
  - (d) Gather a set of instances for experimentation purposes. This was done by selecting some benchmark instances and generating some new instances as well.
  - (e) Develop the instance representation scheme, applying the methodology developed previously in this dissertation.
  - (f) Design and perform several types of experiments, to evaluate the model and to measure the robustness of the algorithm.
  - (g) Analyze the results and draw conclusions.
  - (h) Write an article with the findings and conclusions [100].
6. Perform a low-level heuristic analysis. This is a detailed analysis about another critical aspect of the solution model developed. Now, the focus is on the characteristics of the low-level heuristics. To make this analysis, the hyper-heuristic model developed in the previous step is taken as a basis.
  - (a) When solving instances with a generated hyper-heuristic, perform an analysis about the utilization of every low-level heuristic, in terms of sequence, frequency and time each heuristic is used before changing to another.
  - (b) Write about the findings and conclusions.
7. Work towards a deeper understanding of the Bin Packing Problem structure.
  - (a) Explore some graphical techniques to visually assess similarities and differences among the testbed instances. Principal Component Analysis was the chosen technique. The Self-organizing map, sometimes called a Kohonen map was another method considered in a preliminary phase of this research.
  - (b) Build and analyze some graphs. Distribute feature values and heuristic/hyper-heuristic performances all over the map.

- (c) Draw some conclusions about relations between features and heuristic/hyper-heuristic performance.
  - (d) Write an article with the findings and conclusions.
8. Thesis document. A thesis document that integrates all the research done is written, revised and defended.
- (a) Write the thesis document.
  - (b) Revision of the thesis document.
  - (c) Prepare the dissertation presentation.

A short research leave at Nottingham University with the Automated Scheduling, Optimization and Planning (ASAP) Research Group was done in the Summer of 2010. During this research leave, the main topic of study was the heuristic DJD (point 3 of the methodology) which findings are reported in Chapter 7. Also, a brief study was performed with the objective of discovering homogeneous subsets over a huge set of 1D BPP instances and then, evolve specialized heuristics for each subset. This brief research was supported by the idea that *heuristics can be evolved to be specialists on a particular sub-problem, or general enough to work on all sub-problems. However there is a trade-off between performance and generalization* [34]. After making 8 classes of 300 1D BPP instances with the  $k$ -means clustering algorithm, specialized heuristics were evolved for each class using Genetic Programming. Results were not as good as expected, that is why this experimentation did not go further.

## 4.2 Problem Instances Testbed

Our experimental testbed is comprised of a total of 1418 instances which are summarized in Table 4.1. These instances are classified into three main categories: 1D instances, convex 2D instances and non-convex 2D instances, which are described in next three sections.

Along this large set of experimental instances, there is a huge variety of feature values, beginning with the difference in dimensionality. For example, there are instances whose pieces have an average size of  $1/30$  of the object, while other instances have huge pieces (averaging almost  $2/3$  of object size). The optimum number of objects (or best known results) go from 2 to 373.

### 4.2.1 1D Instances

The 397 one-dimensional problem instances were drawn from the literature. Their characteristics are listed in Table 4.2. Pieces sizes are measured as fraction of one bin. The first eight types of 1D instances are from Scholl et al. [130], where we chose one out of every four instances in Scholl's data bases 1 and 2. Wäscher instances [153] have different number of pieces each<sup>1</sup>. The last four types of 1D instances are triplets from Falkenauer [50] whose optimal solutions have exactly 3 items per bin with zero waste. The instances that are triplets

---

<sup>1</sup>Among the 17 Wäscher instances, obtained from <http://paginas.fe.up.pt/~esicup/tiki-index.php>, there are two instances with the same name, but they have different values, indeed.

Table 4.1: Description of problem instances.

1D			Convex 2D			Non Convex 2D		
Type	num. of instances	num. of pieces	Type	num. of instances	num. of pieces	Type	num. of instances	num. of pieces
DB1 n1	45	50	Conv A	30	30	NConv A	30	35 - 50
DB1 n2	45	100	Conv B	30	30	NConv B	30	40 - 52
DB1 n3	45	200	Conv C	30	36	NConv C	30	42 - 60
DB1 n4	45	500	Conv D	30	60	NConv F	30	35 - 45
DB2 n1	30	50	Conv E	30	60	NConv H	30	42 - 60
DB2 n2	30	100	Conv F	30	30	NConv L	30	35 - 45
DB2 n3	30	200	Conv G	30	36	NConv M	30	45 - 58
DB2 n4	30	500	Conv H	30	36	NConv O	30	33 - 43
Wäscher	17	57 - 239	Conv I	30	60	NConv S	30	17 - 20
Trip60	20	60	Conv J	30	60	NConv T	30	30 - 40
Trip120	20	120	Conv K	30	54	NConv U	30	20 - 33
Trip249	20	249	Conv L	30	30	NConv V	30	15 - 18
Trip501	20	501	Conv M	30	40	NConv W	30	24 - 28
			Conv N	30	60	NConv X	30	25 - 39
			Conv O	30	28	NConv Y	30	40 - 50
			Conv P	30	56	NConv Z	30	60
			Conv Q	30	60			
			Conv R	30	54			
			Fu	1	12			
Total	397		Total	541		Total	480	

originally have item sizes rounded to one decimal place. These instances were scaled to a factor of 10, because our implementation works only with integers for the sizes of the items.

### 4.2.2 Convex 2D Instances

We have 540 two-dimensional instances containing only convex polygonal pieces that were randomly generated in previous work [143] with an algorithm which is a modification of the one used to produce random instances with rectangular pieces before [138]. The generated pieces are convex irregular polygons with a number of sides between 3 and 8. The algorithm starts by generating in a random fashion an initial number of rectangles, specified by a given parameter. The next step is to divide up the rectangles into a number of pieces, until the total number of pieces is completed. The parameters needed to create an instance are the number of objects, their dimensions, the number of pieces in each object, the minimum side in a piece, the maximum ratio between the largest and smallest side (it determines the irregularity or rectangularity factor), and the initial number of rectangles. For more details about this algorithm see [98].

A total of 540 instances were generated within 18 different types. Their characteristics are listed in Table 4.3. This testbed includes 30 rectangular instances (type *Conv I*). All the 2D convex instances, except type *Conv G*, have an optimum with zero waste; that is, in the optimum solution, all objects must be filled up to 100%. Objects for all instances are squares of dimensions  $1000 \times 1000$ . Most of the cases, optimum solution has the same number of pieces per objects, except in types *Conv C*, *Conv K*, *Conv P* and *Conv R* (more details in [98]).

We also added a problem instance called *Fu* from the literature [54]. The instance was scaled by a factor of 10 and was assigned the object size of  $300 \times 300$ , so its dimensions will be more compatible with the dimensions of the generated instances for graphical displays. This instance was included in the investigation reported in Chapters 5 and 6.

Table 4.2: Characteristics of the 1D problem instances

	Average piece size	Piece size standard deviation	Optimal (number of objects)
minimum	0.106	0.011	6
total average	0.359	0.128	81.4
maximum	0.669	0.322	373
Average of instances per type			
DB1 n1	0.485	0.199	26.6
DB1 n2	0.487	0.202	51.8
DB1 n3	0.489	0.203	102.7
DB1 n4	0.488	0.202	254.0
DB2 n1	0.199	0.060	10.5
DB2 n2	0.201	0.062	20.7
DB2 n3	0.199	0.061	40.2
DB2 n4	0.198	0.060	99.8
Wäscher	0.255	0.062	unknown
Trip60	0.333	0.077	20
Trip120	0.333	0.075	40
Trip249	0.333	0.075	83
Trip501	0.333	0.074	167

This instance set contains a wide variety of feature values (see Table 4.3). Among the 2D instances, the average piece size ranges from  $1/30$  to  $1/3$  of the object size and the average rectangularity can be anywhere between 0.35 and 1.0. Rectangularity is a quantity that represents the proportion between the area of a piece and the area of a horizontal rectangle containing it. The lower the rectangularity, the more irregular the pieces are. As one can see in Table 4.3, instances of type *Conv I* have a rectangularity value of 1 which means that all of these instances are rectangular.

### 4.2.3 Non-convex 2D Instances

The 480 new 2D instances containing some non-convex polygons were randomly produced for this investigation with the algorithm described in the next section. The first half of non-convex instances were generated splitting at least five pieces from each instance from types *Conv A*, *Conv B*, *Conv C*, *Conv F*, *Conv H*, *Conv L*, *Conv M* and *Conv O*, respectively. Convex pieces from these instances were randomly selected and then split into two pieces: one convex and one non-convex polygon. The other half of the non-convex instances were produced by creating new convex instances and then splitting some of the pieces into non-convex polygons. Objects for all instances are squares of dimensions  $1000 \times 1000$ . Instances properties are listed in Table 4.4. Pieces sizes are measured as fraction of one object. Concavity degree and ratio (area of piece)/(area of convex hull) are properties explained below.

According to Wang [152], the degree of concavity is defined as the concaveness of the

Table 4.3: Characteristics of the convex 2D problem instances.

	Average piece size	Piece size standard deviation	Average rectan- gularity	Percentage of right angles	Percentage of orthogonal sides	Optimal (number of objects)
minimum	0.033	0.003	0.35	11	34	2
total average	0.154	0.100	0.68	42	65	5.9
maximum	0.354	0.280	1	100	100	15
Average of instances per type						
Conv A	0.100	0.069	0.70	42	68	3
Conv B	0.333	0.162	0.87	67	84	10
Conv C	0.167	0.124	0.68	36	63	6
Conv D	0.050	0.036	0.57	23	51	3
Conv E	0.050	0.035	0.41	12	38	3
Conv F	0.067	0.050	0.59	29	57	2
Conv G	0.332	0.156	0.87	67	83	$\leq 15$
Conv H	0.333	0.158	0.86	67	83	12
Conv I	0.053	0.017	1	100	100	3
Conv J	0.067	0.034	0.83	68	83	4
Conv K	0.154	0.150	0.63	34	60	6
Conv L	0.100	0.075	0.51	23	50	3
Conv M	0.125	0.102	0.55	28	55	5
Conv N	0.033	0.024	0.62	32	60	2
Conv O	0.250	0.223	0.57	27	58	7
Conv P	0.143	0.173	0.49	18	43	8
Conv Q	0.250	0.053	0.89	51	76	15
Conv R	0.167	0.153	0.63	36	62	9
Fu	0.096	0.003	0.76	63	77	2

largest internal angle and it can be computed by  $DC = \frac{B}{A}$  (see Figure 4.1). For 1D items and 2D convex polygons (including rectangles), the degree of concavity is equal to 1. The degree of concavity for a concave polygon is more than one and it is computed with its largest internal angle. The concavities we are dealing with are constructed by a triangle.

The convex hull of a given set  $S$  of points in the plane, is the smallest convex polygon that contains all of the points of  $S$ . The convex hull may be easily visualized by imagining an elastic band stretched open to encompass the given object; when released, it will assume the shape of the required convex hull. The convex hull of a given polygon is defined as the convex hull of all its vertices. The convex hull of a convex polygon is the polygon itself. The convex hull of a non-convex polygon has a greater area than the non-convex polygon (Figure 4.1). So, the relation (area of piece)/(area of convex hull) is less than one only when dealing with non-convex polygons.

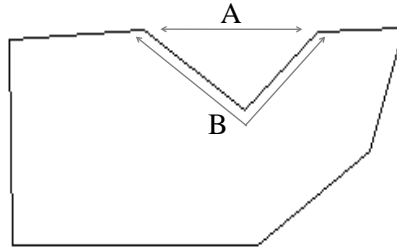


Figure 4.1: Degree of concavity.

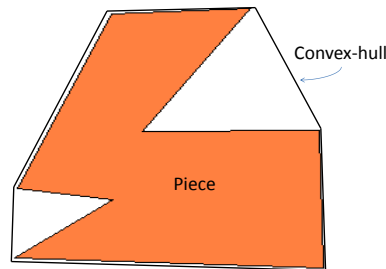


Figure 4.2: A piece and its convex hull.

#### 4.2.4 Algorithm for Producing Random Instances that Include Non-convex Pieces

Our algorithm for producing instances that include non-convex pieces takes as an input a problem instance with convex pieces, then randomly selects some convex polygons and split each one of them into two pieces: one convex and one non-convex polygon. This algorithm was designed to produce the 480 non-convex instances of our experimental testbed. Our implementation requires integers as the pieces coordinates because our solver is programmed this way. Nevertheless, edge lengths are not required to be integers.

We also have designed an algorithm for randomly generate problem instances with convex pieces [143], which can be run to produce the convex instances this algorithm needs. Besides the problem instance as the main input, the following parameters or restrictions are useful to control the irregularity of the new pieces:

- Number of pieces selected to split.
- Minimum length of any edge of the new pieces.
- Maximum internal angle of the new non-convex polygons (this will determine the concaveness).
- Minimum internal angle of any new piece.
- Maximum ratio between the largest and smallest edge of any new piece.

Table 4.4: Characteristics of the non-convex 2D problem instances.

	Average piece size	Piece size standard deviation	Average rectangularity	Percentage of right angles	Percentage of orthogonal sides	Average of concavity degree	Average of ratio area / convex hull	Optimal (number of objects)
minimum	0.044	0.036	0.38	6	27	1.004	0.834	2
total average	0.160	0.135	0.59	26	50	1.13	0.930	5.9
maximum	0.333	0.314	0.84	60	74	1.56	0.987	12
Average of instances per type								
Nconv A	0.074	0.062	0.60	28	52	1.12	0.935	3
Nconv B	0.214	0.158	0.69	38	58	1.22	0.923	10
Nconv C	0.123	0.111	0.59	25	49	1.11	0.939	6
Nconv F	0.051	0.045	0.53	20	46	1.10	0.940	2
Nconv H	0.245	0.163	0.73	46	64	1.15	0.944	12
Nconv L	0.076	0.065	0.47	16	41	1.10	0.941	3
Nconv M	0.099	0.092	0.50	20	46	1.07	0.956	5
Nconv O	0.186	0.190	0.51	19	46	1.10	0.940	7
Nconv S	0.106	0.097	0.45	10	33	1.16	0.918	2
Nconv T	0.293	0.239	0.60	26	51	1.24	0.916	10
Nconv U	0.197	0.161	0.55	17	44	1.19	0.888	5
Nconv V	0.306	0.236	0.62	27	54	1.09	0.936	5
Nconv W	0.155	0.097	0.78	53	69	1.12	0.931	4
Nconv X	0.097	0.072	0.66	32	53	1.17	0.895	3
Nconv Y	0.135	0.129	0.61	25	51	1.09	0.943	6
Nconv Z	0.200	0.234	0.54	19	45	1.09	0.940	12

The last two restrictions will affect the irregularity and rectangularity factor. The outline of the algorithm is as follows:

1. Randomly choose a convex piece to be split and select two points  $Q \neq R$  with integer coordinates somewhere on the boundary of the shape.

This is done by selecting two different edges from the original piece and choosing a point inside each one of them. To ensure that a point on the edge  $E$  with vertices  $(x_1, y_1)$  and  $(x_2, y_2)$  has integer coordinates the following process is done: First, choose an integer value  $x$  in the range  $x_1$  to  $x_2$ , then find the coordinate  $y$  on edge  $E$  with horizontal coordinate  $x$ . If the correspondent  $y$  value is not integer, try again choosing a new  $x$ . When the algorithm has failed 100 times to find a random point with integer coordinates on edge  $E$ , it selects one out of the two vertices of the edge. Finding a point with integer coordinates over an edge may be difficult or impossible for some sloping edges. In such cases, a vertex is selected.

2. Randomly choose a point  $P$  inside the piece.

Here is the process to choose a point inside a shape (Figure 4.3):

- (a) Select a point on a random edge.
- (b) From the selected point draw a ray that crosses the shape in either vertical or horizontal direction. If both directions are possible, select randomly.
- (c) Select a point on the ray such that it is inside the shape.

The cut  $QPR$  separates the original shape into a convex shape and a non-convex shape (Figure 4.4), unless  $P$  falls on the line segment  $QR$ . In this case, the algorithm will choose three new points  $P$ ,  $Q$  and  $R$ .

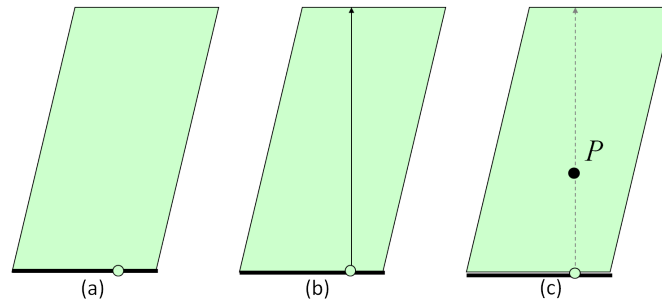


Figure 4.3: Procedure for selecting a point inside a shape.

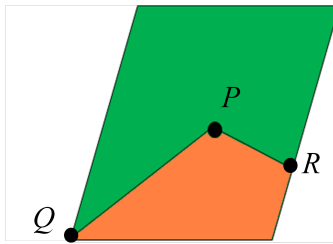


Figure 4.4: One convex and one non-convex polygon created by the developed algorithm.

3. Check that these two new shapes satisfy the desired restrictions, including the fact that we actually obtain a convex and a non-convex shape. If this is the case, replace the original selected piece by the two new pieces. If not, start the algorithm again.

Finally, the algorithm randomizes the order of all the pieces to prevent that the two parts of a split piece end in consecutive positions in the list of pieces. Figure 4.5 shows how a 30-piece instance whose optimum is 3 objects has been transformed into a 35-piece instance with 5 nonconvex pieces using the developed algorithm.

### 4.3 Algorithms Developed for Geometric Computation

Some algorithms for dealing with irregular shapes were developed and implemented for this investigation. The algorithms presented in this section are the building blocks of the placement heuristics (Section 2.6.2). Although most of them are based on basic geometrical concepts, particular cases and exceptions deserve special care. Besides, the easiest-to-solve cases should be reviewed first in order to avoid unnecessary computations. For example, when checking whether a point is inside a shape, a quick computation to determine if the point is above (or below) the top or (the bottom) of the piece will discard many cases. Trivial cases like this one are the most frequent scenario when applying the placement heuristics of this dissertation.

Some considerations to take into account are:

1. All coordinates of every piece are given counterclockwise.
2. The function that reviews if two segments have an intersection returns **false** if they belong to the same line, even if one segment touches the other by one of their ends or if they



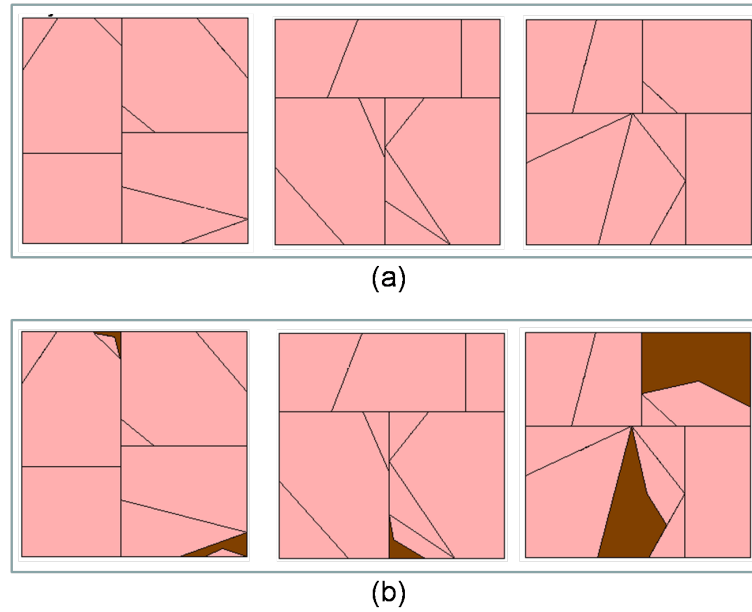


Figure 4.5: (a) A convex problem instance. (b) 5 pieces were randomly selected to build 5 non-convex polygons.

overlap. In other words, our definition of intersection of segments refers to segments that *crosses* but are not coincident. Note that this definition implies that reviewing for intersection of two segments that are exactly the same, the function will return **false** (Note: For reviewing if two shapes intersect, intersection is reviewed for all sides of both shapes. Adjacent sides from non-overlapping adjacent pieces (sharing an edge) are not intersecting).

3. The function that reviews if a point is inside a segment returns **true** if the point is one of the ends of the segment. When the sum of the distances from the point to the two ends of the segment is equal to the segment length, then we consider that the point belongs to the segment.

To know whether two pieces intersects each other, a routine that checks intersection for each pair of sides from both pieces was implemented (Algorithm 1). Initially, a revision is done to confirm that the orthogonal rectangles that circumscribe both pieces intersect. This way we discard the easiest non-intersection cases. This test does not work if one piece is completely inside the other, in which case no edges intersect but the pieces do intersect. That is why this algorithm is always followed by Algorithm 3 that reviews if one piece is completely inside another.

The Algorithm 2 determines whether a point is inside a shape. In case the point is along an edge of the piece or one of its vertices, then the algorithm will return false. The basic idea is to trace a ray from the point to any fixed direction. If the ray cuts the shape an odd number of times, then the point is inside the shape; otherwise it is outside. If the ray touches a vertex of the shape; it is important to determine if the ray touches the shape tangentially or if it actually crosses the shape (see Figure 4.6). This is done employing the  $D$ -function (equation 2.1).

---

**Algorithm 1** Decide if two pieces intersects each other.

---

**Input:** A list of coordinates of two pieces  $P_1$  and  $P_2$ .

**Output:** A boolean value indicating whether the two pieces intersects each other.

```

if lowest end of  $P_1$  is above upper end of  $P_2$  OR lowest end of  $P_2$  is above upper end of  $P_1$ 
then
  return false
if leftmost end of  $P_1$  is right of the rightmost end of  $P_2$  OR leftmost end of  $P_2$  is right of
the rightmost end of  $P_1$  then
  return false

for all edges  $e_1$  of  $P_1$  do
  for all edges  $e_2$  of  $P_2$  do
    if Intersects( $e_1, e_2$ ) then
      return true
return false

```

---

The Algorithm 2 is employed in the procedure to determine is a piece is completely inside another piece which is developed in Algorithm 3. Initially, a revision is done to confirm that the orthogonal rectangles that circumscribe both pieces intersect and the actual pieces do not intersect (part 1). If both pieces do not intersect, we find the orthogonal rectangle that circumscribe both pieces at the same time. If the area of this rectangle is less than the sum of areas of both pieces, it means unequivocally that one piece is inside the another (part 2). If the point in the middle of piece 1 is inside piece 1, then we check if this point is inside piece 2. The same is checked for the middle point of piece 2 (part 3). If this is not the case, then, all vertices and edge midpoints from both pieces are checked to know if they are inside the other piece. Checking vertices and edges midpoints is not an infallible test. It is possible to find a case where all vertices edges midpoints of the inside shape are all along the contour of the larger piece. See for example Figure 4.7. That is why, two points close to each vertex (one for each of the edges) is also tested (part 4). Finally, it is convenient to check whether the two pieces are not equal and in the same position (part 5).

The Algorithm 4 returns the distance in which two segments coincide. This algorithm constitutes the basis for implementing the heuristic called Constructive Approach with Maximum Adjacency in which adjacency is computed between an object with all its placed pieces and a new piece to be placed in several candidate positions.

The algorithm 5 builds a piece that holds all the area in the object that is left of a given piece (see Figure 4.8a). A similar procedure is done to build a piece containing all the area below a given piece (see Figure 4.8b).

The algorithm 6 computes the distance by which a point can reach horizontally a segment. An analogous procedure finds a vertical distance from a point to a segment.

Algorithms 5 and 6 are needed for Algorithm 7 which computes the distance that a given piece can be moved to the left avoiding collision against other pieces in the object and without exceeding the object limits. A similar procedure was implemented in this investigation to find how much a given piece can be moved down. The implementation of this algorithm is basic

---

**Algorithm 2** Decide if a point is inside a shape.

---

**Require:** A list of coordinates of a piece and a point  $(x, y)$ .

**Ensure:** A boolean value indicating whether the point is or not inside the piece.

```

if  $x \leq$  the piece lowest part OR  $x \geq$  the piece upper part then
  return false
if  $y \leq$  the piece leftmost part OR  $y \geq$  the piece rightmost part then
  return false
for all vertices of the piece do
  if the point  $(x, y)$  is equal to the vertex then
    return false
for all sides of the piece do
  if the point  $(x, y)$  is along the side then
    return false

```

Create the point  $(M, y)$ , where  $M$  is a very large number.

```

for all sides of the piece do
  if the side of the piece intersects the segment  $(x, y)$  to  $(M, y)$  then
    counter ++
for all vertices  $i$  of the piece do
  if the vertex belong to the segment  $(x, y)$  to  $(M, y)$  then
     $D_1 \leftarrow Dfunction(\text{segment}, \text{vertex } i - 1)$  [see Equation 2.1]
     $D_2 \leftarrow Dfunction(\text{segment}, \text{vertex } i + 1)$ 
    if  $D_1$  and  $D_2$  have different signs then
      counter ++

if counter is odd then
  return true
else
  return false

```

---

---

**Algorithm 3** Decide if a shape is completely inside another shape.

---

**Require:** The two pieces  $P_1$  and  $P_2$ .

**Ensure:** A boolean value indicating whether one of the pieces is inside the other.

*Part 1*

**if** lowest end of  $P_1$  is above upper end of  $P_2$  **OR** lowest end of  $P_2$  is above upper end of  $P_1$   
**then**

**return false**

**if** leftmost end of  $P_1$  is right of the rightmost end of  $P_2$  **OR** leftmost end of  $P_2$  is right of  
the rightmost end of  $P_1$  **then**

**return false**

**if** the 2 pieces intersect each other **then**

**return false**

    [see Algorithm 1]

*Part 2*

[At this point we only have pieces that do not intersect each other]

$y_{max} \leftarrow \max(\text{maximum } P_1 \text{ y-coordinate, maximum } P_2 \text{ y-coordinate})$

$y_{min} \leftarrow \min(\text{minimum } P_1 \text{ y-coordinate, minimum } P_2 \text{ y-coordinate})$

$x_{max} \leftarrow \max(\text{maximum } P_1 \text{ x-coordinate, maximum } P_2 \text{ x-coordinate})$

$x_{min} \leftarrow \min(\text{minimum } P_1 \text{ x-coordinate, minimum } P_2 \text{ x-coordinate})$

**if**  $(y_{max} - y_{min})(x_{max} - x_{min}) < (\text{area of } P_1 + \text{area of } P_2)$  **then**

**return true**

*Part 3*

$\bar{y}_1 \leftarrow \text{average}(\text{maximum } P_1 \text{ y-coordinate, minimum } P_1 \text{ y-coordinate})$

$\bar{x}_1 \leftarrow \text{average}(\text{maximum } P_1 \text{ x-coordinate, minimum } P_1 \text{ x-coordinate})$

$\bar{y}_2 \leftarrow \text{average}(\text{maximum } P_2 \text{ y-coordinate, minimum } P_2 \text{ y-coordinate})$

$\bar{x}_2 \leftarrow \text{average}(\text{maximum } P_2 \text{ x-coordinate, minimum } P_2 \text{ x-coordinate})$

**if** point  $(\bar{x}_1, \bar{y}_1)$  is inside  $P_1$  and  $P_2$  **or** point  $(\bar{x}_2, \bar{y}_2)$  is inside  $P_1$  and  $P_2$  **then**

**return true**

*Part 4*

**for all** vertices **and** edge midpoints **and** points near each vertex of  $P_1$  **do**

**if** inside  $P_2$  **then**

**return true**

**for all** vertices **and** edge midpoints **and** points near each vertex of  $P_2$  **do**

**if** inside  $P_1$  **then**

**return true**

*Part 5*

**if**  $P_1$  is equal to  $P_2$  **and** in the same position **then**

**return true**

**else**

**return false**

---

---

**Algorithm 4** Measures the distance in which two segments coincide.

---

**Require:** The two finite segments  $S_1$  and  $S_2$ .

**Ensure:** The distance in which  $S_1$  and  $S_2$  coincide.

```

if lowest end of  $S_1$  is above upper end of  $S_2$  OR lowest end of  $S_2$  is above upper end of  $S_1$ 
then
  return 0
if leftmost end of  $S_1$  is right of the rightmost end of  $S_2$  OR leftmost end of  $S_2$  is right of
the rightmost end of  $S_1$  then
  return 0
if slope of  $S_1 \neq$  slope of  $S_2$  then
  return 0
if  $y$ -intercept of  $S_1 \neq$   $y$ -intercept of  $S_2$  then
  return 0 [segments are parallel]
if  $S_1$  and  $S_2$  are both horizontal then
   $p_1 \leftarrow$  rightmost point out of the leftmost ends of  $S_1$  and  $S_2$ .
   $p_2 \leftarrow$  leftmost point out of the rightmost ends of  $S_1$  and  $S_2$ .
  return distance from  $p_1$  to  $p_2$ 
else
   $p_1 \leftarrow$  upper point out of the lowest ends of  $S_1$  and  $S_2$ .
   $p_2 \leftarrow$  lowest point out of the upper ends of  $S_1$  and  $S_2$ .
  return distance from  $p_1$  to  $p_2$ 

```

---



---

**Algorithm 5** Builds a piece containing all the area at the left of a given piece.

---

**Require:** A piece  $P$ .

**Ensure:** A piece whose area is the same that the left area of  $P$ .

Find  $(x_1, y_1)$ , the vertex at the top of  $P$  which is leftmost. [point  $A$  in Figure 4.8a]

Find  $(x_2, y_2)$ , the vertex at the bottom of  $P$  which is leftmost. [point  $D$  in Figure 4.8a]

**return** The piece comprised by the following vertices:

$(x_1, y_1)$

$(0, y_1)$

$(0, y_2)$

$(x_2, y_2)$  and

all vertices in  $P$  between  $(x_2, y_2)$  and  $(x_1, y_1)$

---

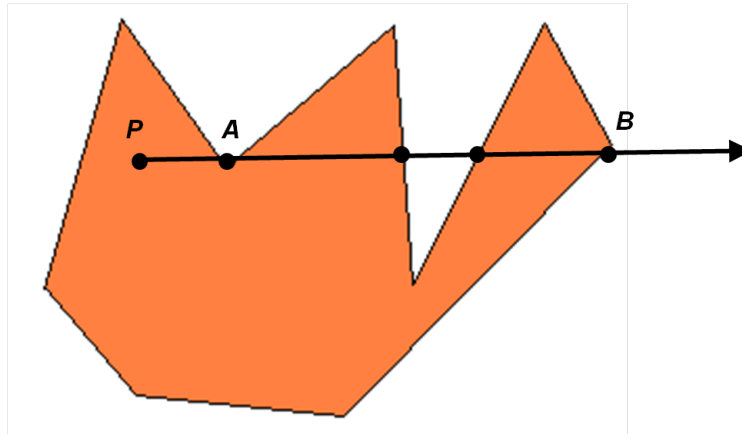


Figure 4.6: Ray from point  $P$  to the right actually touches 4 times the shape boundaries. The ray crosses the shape at vertex  $B$ . In contrast, the ray touches vertex  $A$  only tangentially and does not cross the shape at this point. Therefore, the count for crosses is 3. Since 3 is an odd number, we conclude that  $P$  is inside the shape.

for bottom-left moves that take place in all placement heuristics.

## 4.4 Summary

This chapter presented some methodological issues related with this dissertation. Testbed instances were described here and they are employed in the experiments presented in the following five chapters. Some geometric algorithms are described as well. The next chapter presents an implementation of the solution model for the 2D irregular BPP that includes only convex polygons.

---

**Algorithm 6** Computes the horizontal distance from a point to a given segment. Distance is zero if the point is along the segment. Distance is positive if the point is in the right of the segment. Otherwise it is negative.

---

**Require:** A point  $(x, y)$  and a segment defined by points  $(x_1, y_1)$  and  $(x_2, y_2)$ .

**Ensure:** The horizontal distance from  $(x, y)$  to the segment defined by  $(x_1, y_1)$  and  $(x_2, y_2)$ .

```

if  $(y < y_1$  and  $y < y_2)$  or  $(y > y_1$  and  $y > y_2)$  then
  return ‘The point does not reach horizontally the segment’
if  $(y = y_1$  and  $y = y_2)$  and  $(x > x_1$  and  $x > x_2)$  then
  return  $\min(x - x_1, x - x_2)$ 
if  $(y = y_1$  and  $y = y_2)$  and  $(x < x_1$  and  $x < x_2)$  then
  return  $-\min(x_1 - x, x_2 - x)$ 
if  $(y = y_1$  and  $y = y_2)$  then
  return 0
else
  return  $x - x_1 + (x_1 - x_2)(y_1 - y)/(y_1 - y_2)$ 

```

---



---

**Algorithm 7** Computes the distance that a given piece can be moved to the left without overlapping other pieces and without exceeding the object limits.

---

**Require:** A piece  $P$  and the other pieces that are inside the same object.

**Ensure:** The distance that  $P$  can be moved to the left.

Build piece  $P'$  whose area is the same that area at the left of  $P$  [Algorithm 5 and Figure 4.8a].

Find the set  $S$  containing all pieces in the object that intersect or are inside  $P'$  but do not intersect nor are inside  $P$ .

$m \leftarrow$  minimum  $x$ -coordinate of  $P$

**if**  $S$  is empty **then**

**return**  $m$

**for all** vertices  $i$  of  $P$  **do**

**for all** edges  $j$  of **all** pieces in the object **do**

**if** vertex  $i$  reaches edge  $j$  projecting to the left **then**

$d \leftarrow$  distance from vertex  $i$  to edge  $j$  [Algorithm 6]

**if**  $d < m$  **then**

$m \leftarrow d$

**return**  $m$

---

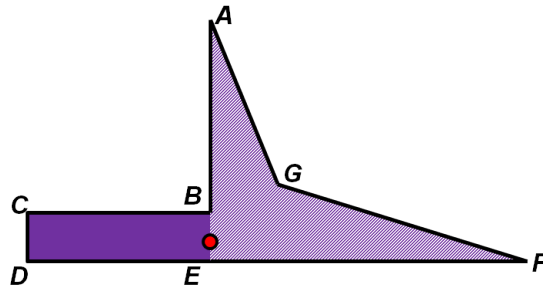


Figure 4.7: Piece  $AEFG$  is inside piece  $ABCDEFG$ . In this case, checking if all vertices and edges midpoints of  $AEFG$  are inside  $ABCDEFG$  will return **false**. Only when a point very close to vertex  $E$  is found inside  $ABCDEFG$ , the algorithm returns **true** to the question about if one of the pieces is inside the other. In this case, reviewing intersection of these two pieces with Algorithm 1 will return **false** because none of the sides crosses another (although they coincide).

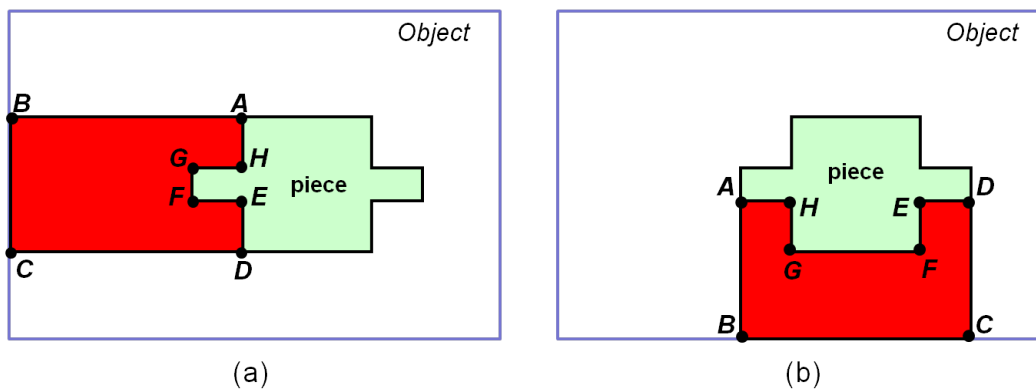


Figure 4.8: Piece  $ABCDEFGH$  contains all area to the (a) left and (b) below a given piece.



# Chapter 5

## Hyper-heuristics for 2D Irregular BPP (Convex)

In this chapter we describe how the developed model explained in Chapter 3 was applied to the 2D Irregular BPP with convex polygons [98, 143]. Results from this chapter may be taken as preliminary, since research presented in the Chapters 6, 7 and 8 make significant improvements to the main elements of the solution model.

### 5.1 Experimental Setup

This section describes how the elements of the evolutionary framework were implemented in this first implementation.

#### 5.1.1 Set of heuristics

In a one-dimensional packing problem, the related heuristics refer to the way the pieces are selected and the bins in which they will be packed. For a two-dimensional problem (regular and irregular), additional difficulty is introduced by defining the exact location of the figures, that is, where a particular figure should be placed inside the object. That is why, in this investigation two kinds of heuristics were considered: one kind for selecting the figures and objects, and the other for placing the figures into the objects. The ten selection heuristics used are detailed in section 2.6.1. The four placement heuristics utilized are based on the bottom-left heuristics and the constructive approach presented by Hifi and M'Hallah [70]. The full description is in section 2.6.2. Some of these heuristics were taken from the literature, others were adapted, and some other variations were developed.

#### 5.1.2 Chromosomes

Each chromosome in the GA (Section 3.5) is composed by several blocks and each block had nine numbers in this particular implementation. The label (action) is the ninth number, which identifies a particular pairing of a selection heuristic and a placement heuristic. The action was selected from all possible combinations of selection and placement heuristics. There are

40 combinations shown in Table 5.1. This hyper-heuristic generation routine was performed considering the possibility of rotating the items 0, 90, 180 or 270 degrees.

The objective function employed in this implementation is computed as explained in Section 3.4.

### 5.1.3 Representation of problem instance state

For this irregular case, the first three numbers out of the eight that summarizes the problem state are related to rectangularity, a quantity that represents the proportion between the area of a piece and the area of a horizontal rectangle containing it. The first number represents the fraction of remaining pieces with high rectangularity, in the range of 0.9 to 1 inclusive. The second corresponds to those of medium rectangularity, in the range 0.5 to 0.9. The third corresponds to low rectangularity, from 0 to 0.5. The fourth and fifth numbers are related to the area of pieces, indicating the fraction of large and small pieces respectively. Large items are those whose area is larger than  $1/4$  of the object. Small pieces are those larger than  $1/10$  but less or equal to  $1/4$  of the object total area. Fraction of pieces whose area is less or equal to  $1/10$  is not represented in the problem state. The sixth and seventh numbers are related to the height of pieces, indicating the fraction of tall and short pieces respectively. Tall items are those whose vertical dimension is longer than the half of the vertical dimension of the object. Short pieces are those taller than  $1/4$  but less or equal to  $1/2$  of the object total height. The eighth number represents the fraction of the total items that remain to be packed. The features selected for representing instances states are listed in Table 6.1 in Chapter 6.

### 5.1.4 Testbed Instances

The problem instances considered are described in section 4.2.2. Instance characteristics are listed in Tables 4.1 and 4.3.

## 5.2 Experiments

All problem instances (541) were solved with the 40 combinations of selection and placement heuristics. For each instance, the best heuristic, its fitness and the number of objects are recorded. Then, the GA process (Figure 3.1) is run to generate hyper-heuristics.

In order to test the overall performance of the model when tackling irregular problems, various experiments were designed:

- **Experiment I.-** First, instances are divided into two groups: training and testing sets. Training set is formed of instance types *Conv A* to *Conv I* plus the *Fu* instance totaling 271 instances. The best individual generated by the evolving process within the proposed model is the hyper-heuristic chosen to run the rest of the instances comprising the testing set (instance types *Conv J* to *Conv R*). To validate the consistency of the model, two complete and independent runs were performed, from which hyper-heuristics HH1a and HH2b were obtained. Each was tested with the testing set and then compared with those results obtained by each of the 40 single heuristics (each is a combination of selection and placement heuristic) in the same set.

Table 5.1: Representation of actions.

Action	Selection	Placement
1	First Fit (FF)	BLI - Bottom Left (Irregular)
2		CA - Constructive
3		CAA - Constructive - Minimum Area
4		CAD - Constructive - Maximum Adjacency
5	First Fit Decreasing (FFD)	BLI - Bottom Left (Irregular)
6		CA - Constructive
7		CAA - Constructive - Minimum Area
8		CAD - Constructive - Maximum Adjacency
9	First Fit Increasing (FFI)	BLI - Bottom Left (Irregular)
10		CA - Constructive
11		CAA - Constructive - Minimum Area
12		CAD - Constructive - Maximum Adjacency
13	Filler + FFD	BLI - Bottom Left (Irregular)
14		CA - Constructive
15		CAA - Constructive - Minimum Area
16		CAD - Constructive - Maximum Adjacency
17	Next Fit (NF)	BLI - Bottom Left (Irregular)
18		CA - Constructive
19		CAA - Constructive - Minimum Area
20		CAD - Constructive - Maximum Adjacency
21	Next Fit Decreasing (NFD)	BLI - Bottom Left (Irregular)
22		CA - Constructive
23		CAA - Constructive - Minimum Area
24		CAD - Constructive - Maximum Adjacency
25	Best Fit (BF)	BLI - Bottom Left (Irregular)
26		CA - Constructive
27		CAA - Constructive - Minimum Area
28		CAD - Constructive - Maximum Adjacency
29	Best Fit Decreasing (BFD)	BLI - Bottom Left (Irregular)
30		CA - Constructive
31		CAA - Constructive - Minimum Area
32		CAD - Constructive - Maximum Adjacency
33	Worst Fit (WF)	BLI - Bottom Left (Irregular)
34		CA - Constructive
35		CAA - Constructive - Minimum Area
36		CAD - Constructive - Maximum Adjacency
37	Djang and Finch (DJD)	BLI - Bottom Left (Irregular)
38		CA - Constructive
39		CAA - Constructive - Minimum Area
40		CAD - Constructive - Maximum Adjacency

- **Experiment II.-** This experiment is similar to Experiment Type I, except that the training and testing sets are interchanged.
- **Experiment III.-** This experiment takes the *Fu* instance and 15 instances from each problem type (from *Conv A* to *Conv R*) to form the training set with 271 instances. The remaining instances form the testing set.
- **Experiment IV.-** It is the same as Experiment Type III, except that the training and testing sets are swapped.

For each of the experiments II, III and IV one hyper-heuristic was generated. Experiments were conducted with population size of 100, crossover probability of 1.0, mutation probability of 0.05, for 500 generations [143]. These parameters are very close to the chosen in the implementation of the model for the regular case of the 2D BPP [52, 138].

### 5.3 Results and Discussion

Tables 5.2 to 5.5 presents the main results when solving testing sets with the hyper-heuristics generated in each of the experiments, as reported in [143]<sup>1</sup>. Figures in cells indicate the percentage of problems solved that employs a particular number of extra objects (left column) when compared against results provided by the best single heuristic for each case. Figures in last 10 columns average the performance of the four combinations of heuristics for which the selection heuristic is in common.

The resulting hyper-heuristics in Experiment I show better results than those produced by the single heuristics. Hyper-heuristic HHa solved 87.0% with the same number of objects as the best single heuristic. In the remaining 13% of the instances the hyper-heuristic required just one additional object. The closest combination is *DJD* with 43.5%. Moreover, in a good percentage of cases all the single heuristics use more than one object, and in fact, all of them have instances using more than five additional objects. The performance in both hyper-heuristics is very similar, despite the fact that they are formed of rather different blocks. In general, the training instances in this experiment seem to be less irregular than those in the testing set (as it can be inferred from Table 4.3).

Experiment II is the opposite to Experiment I. In other words, the training set in this experiment is composed of more irregular pieces, and then tested with less irregular pieces. Table 5.3 shows results for this experiment. It can be observed that the hyper-heuristic produced solves 86.7% of the instances with the same number of objects. For 0.4% of the cases, only one instance, it needs one object less.

For Experiments III and IV, the training and testing sets had similarities in their instances given that the problem types were evenly divided, leaving half of the instances in the training set, and the other half in the testing set. This explains, in average, a very similar performance in the hyper-heuristics produced by these experiments. The results are shown in Tables 5.4 and 5.5.

<sup>1</sup>Results for single heuristics in Tables 5.2 to 5.5 were recomputed after fixing a bug in the programming code for the selection heuristics. Therefore, results in this document differ from those in previous work reported in [143]. The original programming code for selection heuristics was originally taken from the code used in the work [138].

Table 5.2: Number of extra objects for the testing set and compared against results of the best single heuristics. Experiment I, for first and second independent runs. Figures in last 10 columns average the performance of four heuristics. These heuristics are indicated by the numbers under the selection heuristic name (see Table 5.1).

Extra Obj.	HHa	HHb	Selection Heuristics									
			FF	FFD	FFI	Filler	NF	NFD	BF	BFD	WF	DJD
			1-4	5-8	9-12	13-16	17-20	21-24	25-28	29-32	33-36	37-40
0	87.0	90.4	10.2	38.4	0.3	39.2	1.4	4.0	9.4	39.4	3.1	43.5
1	13.0	9.6	25.7	31.9	20.7	31.8	14.2	13.4	25.7	30.9	20.6	30.7
2			24.1	11.9	14.0	12.2	8.9	12.5	23.2	12.1	18.8	11.2
3			16.0	7.9	17.0	8.1	9.7	11.0	15.6	7.5	18.1	6.9
4			10.0	6.6	16.0	5.4	9.9	11.7	11.6	5.9	14.9	4.3
5			5.9	1.1	11.5	1.1	8.2	9.5	5.6	1.9	9.4	1.6
> 5			8.1	2.2	20.5	2.2	47.7	37.9	8.8	2.1	15.3	1.8

Table 5.3: Number of extra objects for the testing set and compared against results of the best single heuristics. Experiment II. Figures in last 10 columns average the performance of four heuristics. These heuristics are indicated by the numbers under the selection heuristic name (see Table 5.1).

Extra Obj.	HH	Selection Heuristics									
		FF	FFD	FFI	Filler	NF	NFD	BF	BFD	WF	DJD
		1-4	5-8	9-12	13-16	17-20	21-24	25-28	29-32	33-36	37-40
-1	0.4										
0	86.7	14.9	41.0	7.7	40.8	7.5	9.9	14.9	41.0	8.9	53.8
1	10.0	34.2	25.9	24.6	26.3	19.5	21.5	33.7	26.3	26.8	24.7
2	2.6	15.0	15.7	19.6	15.0	15.5	14.4	16.5	15.3	18.2	12.2
3	0.4	15.2	10.2	6.9	10.1	8.3	7.5	15.8	10.1	14.4	6.5
4		14.4	5.3	13.5	5.5	6.2	7.1	12.4	5.4	15.7	2.6
5		4.9	1.3	16	1.6	6.1	5.2	5.3	1.3	11.5	0.2
> 5		1.3	0.6	11.8	0.6	37.0	34.5	1.6	0.6	4.5	

Table 5.4: Number of extra objects for the testing set and compared against results of the best single heuristics. Experiment III. Figures in last 10 columns average the performance of four heuristics. These heuristics are indicated by the numbers under the selection heuristic name (see Table 5.1).

Extra Obj.	HH	Selection Heuristics									
		FF	FFD	FFI	Filler	NF	NFD	BF	BFD	WF	DJD
		1-4	5-8	9-12	13-16	17-20	21-24	25-28	29-32	33-36	37-40
0	91.1	12.2	40.6	3.8	40.3	4.4	6.8	11.9	40.4	5.8	47.9
1	7.8	29.8	26.6	22.3	27.5	17.5	17.6	29.4	27.5	24.1	28.1
2	1.1	19.6	15.0	16.6	14.9	12.1	13.1	19.4	14.5	18.0	12.3
3		15.4	8.9	10.9	8.7	8.2	8.5	16.0	8.6	15.8	6.4
4		13.2	5.8	16.5	5.2	8.0	9.9	13.0	5.6	16.1	3.4
5		5.7	1.5	14.2	1.9	7.3	7.6	6.0	1.9	11.0	1.0
> 5		4.0	1.6	15.7	1.6	42.5	36.6	4.3	1.5	9.2	0.9

Table 5.5: Number of extra objects for the testing set and compared against the best results of the best single heuristics. Experiment IV. Figures in last 10 columns average the performance of four heuristics. These heuristics are indicated by the numbers under the selection heuristic name (see Table 5.1).

Extra Obj.	HH	Selection Heuristics									
		FF	FFD	FFI	Filler	NF	NFD	BF	BFD	WF	DJD
		1-4	5-8	9-12	13-16	17-20	21-24	25-28	29-32	33-36	37-40
0	87.8	12.9	38.7	4.2	39.7	4.5	7.1	12.5	40.0	6.1	49.4
1	10.7	30.2	31.2	23.1	30.5	16.1	17.3	30.0	29.7	23.3	27.4
2	1.1	19.5	12.6	17.0	12.4	12.3	13.8	20.3	12.9	19.0	11.1
3		15.9	9.2	13.0	9.6	9.8	10.0	15.3	9.0	16.6	7.1
4		11.2	6.0	13.0	5.7	8.1	8.9	11.0	5.7	14.5	3.4
5		5.1	0.9	13.3	0.8	7.0	7.1	4.9	1.3	9.9	0.7
> 5		5.4	1.3	16.5	1.3	42.2	35.8	6.1	1.3	10.6	0.8

Table 5.6 summarizes with a single value, the performance of each hyper-heuristic for each experiment carried out. On average, using the hyper-heuristic reduces by more than one object the solution given by the average heuristic for each instance. The hyper-heuristics produced in Experiment I were the ones showing the best performance. Results for experiment II are not as good as the previous experiment. It seems that training with irregular instances first, and then testing with more rectangular problems is not so advantageous.

Table 5.6: Number of extra objects used by the hyper-heuristics in the testing set with respect to the average obtained by the 40 single heuristics.

<b>Extra Objects</b>	
Experiment I, 1st	-1.63
Experiment I, 2nd	-1.66
Experiment II	-1.11
Experiment III	-1.41
Experiment IV	-1.37

Looking at the results it is clear in all cases that the method to create hyper-heuristics and the hyper-heuristics themselves are efficient, at least with respect to the number of objects used for each instance. The GA-based procedure has found hyper-heuristics composed of a set of rules which associate the problem state to a combination of selection and placement heuristics.

## 5.4 Summary

In this chapter, the implementation of the solution model is applied to the 2D irregular BPP (convex). It is important to get a better feeling of the real advantages or the proposed approach, and the practical implications of using it. For example, the computational cost of applying a generated hyper-heuristic to a problem is only slightly higher than the time used by any of the single heuristics, which run in just a few seconds. However, the performance of the model may be improved with some changes. This is done in the following chapters.

In particular in the next chapter, attention is paid to the representation of instance states (explained in Section 3.2) in order to reduce the domain knowledge required to produce a good representation scheme.





## Chapter 6

# Defining a Problem-State Representation Scheme

One interpretation of hyper-heuristic is based on the idea of a high level heuristic that should decide which single low-level heuristic to apply, depending on the given problem state which is summarized by a numerical vector called *representation* [126]. This idea of hyper-heuristic has been applied for solving the 1D and 2D Bin Packing Problem (BPP) [128, 129, 138, 139, 143], as well as the Constraint Satisfaction Problem (CSP) [141]. Other approaches have used representation of instances (and/or instance-states) based on selecting some relevant features. For example, the hyper-heuristic model that uses Case-based Reasoning for Course Timetabling Problems [113].

This chapter focuses on the way of selecting relevant features for a meaningful representation scheme. Much of the performance of a hyper-heuristic model may depend on the choice of representation of the problem state and the choice of the particular set of heuristics used [126]. Smith-Miles et al. [135] showed the dependence of algorithm performance on features measures. They employed the Early/Tardy Machine Scheduling Problem to find that some features can predict heuristic performance and that some other features do not seem to influence heuristic performance at all. In this chapter, we propose a new general methodology for feature selection using data mining. Then, we tested the generated methodology producing a new representation scheme for the 2D irregular BPP which is solved by our hyper-heuristic approach (presented in Chapter 3). The basic assumption is that similar problem states would be associated with the same single heuristic. The objective is to characterize instances into *homogeneous* classes and relate them with the best single heuristic for each class. The idea behind hyper-heuristics is to associate each heuristic with the problem conditions under which it flourishes and hence apply different heuristics to different parts or phases of the solution process. That is why this work attempts to find the key features of a problem that makes instances suitable for one heuristic or another.

For the 2D irregular BPP, a hyper-heuristic solution model has been applied with encouraging results (Chapter 5) in which some of the features selected for the state representation were taken from the 2D regular (rectangular) BPP [138] whose representation was influenced for the one-dimensional case as well [128]. Problems known to be hard have certain characteristics [37]. In 1D BPP, for example, the hard benchmark problems involve items whose weights are typically a significant fraction of the bin capacity, for example at least 20% of bin

Table 6.1: Representation 1 of the instance state.

Feature	Description
<i>Fraction of remaining pieces in the instance:</i>	
1	with high rectangularity, in the range $(0.9, 1]$ .
2	with medium rectangularity, in the range $(0.5, 0.9]$ .
3	with low rectangularity, in the range $(0, 0.5]$ .
4	whose area is larger than $1/4$ of the object area.
5	whose area is larger than 0.1 and up to 0.25 of the object area.
6	taller than the half of the object area.
7	taller than 0.25 and up to 0.5 of the object total height.
8	that remain to be packed.

capacity, so that there will be no more than five items per bin but there will be a very large number of items so that the difficulty arises when trying to find the subsets of items that are to reside in each bin. If there were many very small items, those items could be used essentially as sand to fill up space wasted when large items were packed [126]. This makes much sense in the 1D-BBP. In the 2D case, not only the size is important but also the shape; and in the 2D irregular BPP, some features linked with irregularity appear. Some of these features may be relevant in the representation of the problem state, some others may not. Which features are the most appealing for the representation of the problem state may not be obvious.

The solution model from Chapter 3 and its particular implementation from Chapter 5 are taken as experimental environment. In that chapter, eight features related to the problem domain were selected by intuition to represent an instance state (see Table 6.1). This is called Representation 1 in this chapter research. The features were selected by analyzing various parameters present in irregular 2D pieces, such as rectangularity, area, length, height and the percentage of remaining pieces to be packed. Rectangularity is a quantity that represents the proportion between the area of a piece and the area of a horizontal rectangle containing it.

The main part of this chapter is Section 6.1 which presents the description of a proposed methodology for feature selection. Section 6.2 applies the proposed methodology to develop a representation scheme for the 2D irregular BPP, which is called Representation 2. At this point, preliminary experiments lead us to revise some parameters of the GA. Therefore, we performed again the experiments with the Representation 1, to make the results comparable. The GA parameters were kept equal for experiments with both representations. The only difference was the representation employed. Sections 6.3 and 6.4 presents the experimental evaluation and the discussion of results.

## 6.1 Methodology for Developing a Representation Scheme

This section explains the proposed methodology for developing a representation scheme which uses data mining techniques to determine the feature set. Data mining is the process of finding hidden patterns in large amounts of data. One of its main applications has been scientific discovery. The proposed methodology is then applied to the 2D irregular BPP in next section.

The proposed methodology comprises six steps:

**Step 1.** Solve each instance with each single heuristic from the heuristic repository  $H$  and compute a measure of performance.

**Step 2.** Register the performance of all heuristics for each instance as a vector in  $\mathfrak{R}^{|H|}$ , where  $|H|$  is the size of the heuristics repository  $H$ .

For a given instance,  $\mathbf{q}$  is the vector of performance for the  $|H|$  heuristics,

$$\mathbf{q} = [ q_1 \quad q_2 \quad \cdots \quad q_{|H|} ] \quad (6.1)$$

whose length is given by the square root of the dot product of a vector by itself:

$$|\mathbf{q}| = \sqrt{q_1^2 + \cdots + q_{|H|}^2} \quad (6.2)$$

Normalize this vector, dividing by its length (to get length of one). Normalized performance  $\mathbf{q}^*$  is given by

$$\mathbf{q}^* = \frac{\mathbf{q}}{|\mathbf{q}|} \quad (6.3)$$

Note that  $\mathbf{q}^*$  is a unit vector. Unit vectors are used to indicate direction. They aim in the direction of the best heuristic for each instance (see Figure 6.1).

Normalized performance removes the effect of *easy* or *hard* instances in the measure of performance. For the following example and explanation, an *easy* instance will be an instance that obtains high scores according to the defined measure of performance, while a *hard* instance is the one that obtains lower scores. Note that this practical working definition differs from the concept of hardness in computational complexity. Nevertheless, we found the necessity of looking for terms that describe instances with high or low performance.

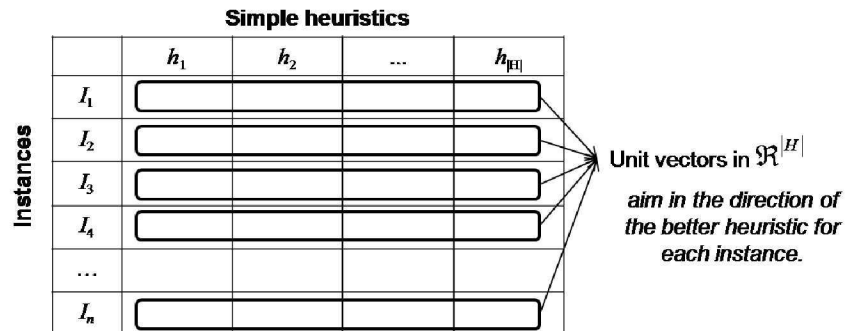


Figure 6.1: Matrix of normalized performance.

**Example.** Let  $\mathbf{q}_a$  and  $\mathbf{q}_b$  be the performance vector for two given instances evaluated with  $H = 6$  heuristics. Let

$\mathbf{q}_a = [ 0.906 \ 0.906 \ 1.000 \ 0.918 \ 0.918 \ 0.922 ]$  with length  $|\mathbf{q}_a| = 2.275$ , and  
 $\mathbf{q}_b = [ 0.604 \ 0.614 \ 0.613 \ 0.616 \ 0.619 \ 0.617 ]$  with length  $|\mathbf{q}_b| = 1.503$ .

The third heuristic is the best for instance  $I_a$  as it produces the maximum performance (1.000), meaning that all objects could be filled at 100%. For instance  $I_b$ , the fifth heuristic is the best (performance = 0.619), following by the sixth heuristic (performance = 0.617). In this example, all six heuristics produce higher performance in  $I_a$  compared to  $I_b$ . For these heuristics, instance  $I_a$  is easier than instance  $I_b$ . The vectors of normalized performance are:

$\mathbf{q}_a^* = \frac{\mathbf{q}_a}{|\mathbf{q}_a|} = [ 0.398 \ 0.398 \ 0.439 \ 0.403 \ 0.403 \ 0.405 ]$ , and  
 $\mathbf{q}_b^* = \frac{\mathbf{q}_b}{|\mathbf{q}_b|} = [ 0.402 \ 0.408 \ 0.408 \ 0.410 \ 0.412 \ 0.410 ]$ .

With vectors  $\mathbf{q}_a^*$  and  $\mathbf{q}_b^*$  we still can see which heuristics produce the best and worst results for a particular instance, but we cannot distinguish which instance produces the higher absolute performance.

The normalized performance of a heuristic in instance  $I_a$  is relative because it depends on the performance of the other heuristics in  $H$  in the same instance  $I_a$ .

It is a good idea to normalize because of the following explanation: The **unnormalized** vector of performance is absolute: for an *easy* instance, most heuristics may have high performance; for a *hard* instance, most heuristics may have low performance. If we compare the unnormalized (absolute) performance of instances  $I_a$  and  $I_b$  when solved by heuristic  $h_i$ , we can see which of the instances is better solved by heuristic  $h_i$ . As in the example above, let's suppose  $I_a$  is much easier than  $I_b$  so that every heuristic in  $H$  has a better performance in  $I_a$  than in  $I_b$ . The higher performance of a particular heuristic  $h_i$  in  $I_a$  compared with  $I_b$  does not tell us nothing about the particular ability of  $h_i$  in instances  $I_a$  and  $I_b$ . But, if we compare the **normalized** performance of  $h_i$  in  $I_a$  and  $I_b$ , we can see in which instance heuristic  $h_i$  does better compared with what the other heuristics can do in the same instances. In conclusion, if we know that normalized performance of  $h_i$  is higher in  $I_a$  than in any other instance, we also know  $h_i$  is particular capable in instance  $I_a$ . This methodology is about finding those features that make heuristics particular capable of solving some kinds of instances.

**Step 3.** Classify all instances into homogeneous clusters according to their normalized performance measure for all heuristics. The number of clusters is a parameter to be chosen. There are many clustering techniques that are based on statistical and AI techniques. One of these is the  $k$ -means technique which is a widely used clustering algorithm known for its observed speed and its simplicity [8]. Given a set of  $n$  data points, the algorithm uses a local search approach to partition the points into  $k$  clusters,  $k < n$ . A set of  $k$  initial cluster centers is chosen arbitrarily. Each point is then assigned to the center closest to it, and the centers are recomputed as centers of mass of their assigned points. This is repeated until the process stabilizes. It can be shown that no partition occurs twice during the course of the algorithm, and so the algorithm is guaranteed to terminate [8]. The objective is to minimize total intra-cluster variance, or, the squared error function.

$$V = \sum_{i=1}^k \sum_{x_j \in S_i} (x_j - \mu_i)^2 \quad (6.4)$$

where there are  $k$  clusters  $S_i$ ,  $i = 1, 2, \dots, k$ , and  $\mu_i$  is the centroid or mean point of all the points  $x_j \in S_i$ . Chiang and Mirkin [38] reviewed several approaches for selecting the *right* number of clusters for the  $k$ -means technique.

There are other alternatives for grouping instances, Jain et al. [80] present a taxonomy of clustering techniques.

**Step 4.** Determine a set of problem features or measures that may be relevant to the ability of the heuristics to solve each instance. For example, in BPP, average size of pieces may be related on how hard an instance is. Next steps will prune the list to keep just the more relevant features.

**Step 5.** Determine pairs or sets of equivalent features. It may happen that some couple or small sets of features are highly correlated (positive or negative) since they carry almost the same information. In this case, we can say we have equivalent features. The Pearson correlation coefficient [125] measures linear dependence between a pair of variables and it is an immediate way to perform this (although other measures of association exist [83]). For example, in the 2D irregular BPP *average area of pieces* and *percentage of small pieces* may be strong and negatively correlated. This can happen for pairs or even for small sets of features where all features in the set are highly correlated with all the others. For every pair or set of equivalent features it is possible to choose just one of them which would *act* as the representative of the other(s) and *delete* the others reducing the total set of features.

**Step 6.** Among the set of the problem features, select a subset which better predicts the instances clusters. This selection could be done through Multinomial Logistic Regression (MLR) [58]. MLR is a statistical method that extends the (binary) logistic regression when the categorical dependent outcome has more than two levels. This method can be used to assess the influence of explanatory variables (problem features) on the nominal response variable (cluster). The MLR model has to be run with all the features as independent variables. Those variables with lower  $p$ -values will be the most significant ones. That is, the features which better predict. Since instances in each cluster tend to have better performance in the same single heuristics, those features that are more related with the clusters will be the same features that influence instances to be better solved for one heuristic or another.

Previous step is especially important if prediction is going to be by MLR, because high correlation among independent variables, called multicollinearity, is undesirable because it makes unstable the model coefficient estimates. In 2008 appeared an application of binary logistic regression in the hyper-heuristic search process [32] where Burke et al. used this technique as an aid to predict if the incoming heuristic (or sequence of heuristics) should not perform well (or even generate an infeasible solution) saving the computational time of calculating the exact objective value of the resulting solution. As far as we know, this is the only application of logistic regression in hyper-heuristic research, but it is done for a totally different purpose. In this report, Burke et al. use another data mining technique for classification and prediction, namely the Multi-Layer Perception (MLP) neural network performing slightly better than the binary logistic regression model.

Figure 6.2 summarizes this procedure as it relates features with heuristic performance with clustering as an intermediate phase.

The features selected by the last step are the ones most suitable to be used in the problem representation.

However, this approach may present the following drawbacks: (1) every instance is solved by the same heuristic from start to end. Though, this approach does not evaluate the performance of heuristics solving partial states of instances. For example, an heuristic could be bad for solving a whole instance but could be very good for placing the last pieces. And,

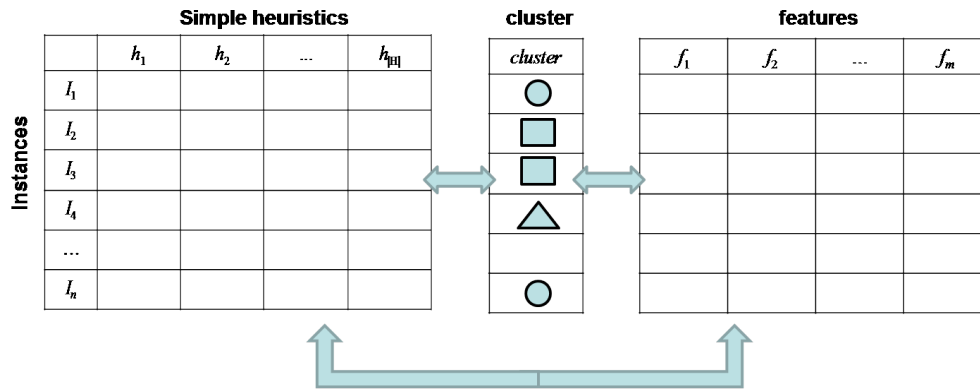


Figure 6.2: General idea for identifying those features related with heuristics performance. First, instances are clustered according to heuristic performance and then features are related with cluster membership.

(2) the developed approach does not measure the impact of successive application of different single heuristics. There might be some heuristic that performs badly on many problems but happens to be vital for obtaining good solutions for a small number of problems, but only when used in combination with some other heuristic as well. For tackling these drawbacks, the evolutionary process may play an important role. That is why, this is the process that constructs the hyper-heuristic. In our model, is the GA the one that would discover if some single heuristics are good only for some sort of instances states. Also, we would expect that the GA evaluate better those hyper-heuristics which involve successful combination of heuristics. In conclusion, we can consider the methodology described in this section as an aid for finding out good features to use in the problem state representation for the evolutionary model.

## 6.2 Developing a New Representation Scheme

In order to test the data-mining based methodology, the 2D irregular BPP is chosen. We applied the six steps from previous section in the same 541 instances and 40 single heuristics used in [143] (which are described in Chapter 5). In our particular experimental environment, the performance measure of each heuristic over each instance was computed with equation 3.2, which ranges from values greater than 0 and up to 1. Heuristics performance for each instance was seen as a vector in a 40-dimensional space. Every vector was normalized. We applied the  $k$ -means algorithm for clustering the 541 available instances into five groups according to their normalized heuristic performance.

We applied the  $k$ -means algorithm for clustering the 541 available instances of the 2D irregular BPP into five homogeneous groups according to their normalized heuristic performance. The number of instances in each cluster is: 61, 124, 165, 144 and 47. The algorithm was stopped after the 50<sup>th</sup> iteration<sup>1</sup>.

Nineteen numerical features were computed for each instance:

<sup>1</sup>The  $k$ -means algorithm was performed in the software SPSS for Windows, release 10.0.1 (<http://www.spss.com/>).

1. Number of pieces.
2. Mean number of sides for the instance pieces.
3. Variance of the number of sides of all instance pieces.
4. Mean area for the instance pieces (area for each piece is measure in fraction of the object total area).
5. Variance of the area of all instance pieces.
6. Mean height for the instance pieces (height for each piece is measured as a fraction of the object height with the difference between its maximum and minimum  $y$  coordinates).
7. Variance of the height of all instance pieces.
8. Mean width for the instance pieces (width for each piece is measured as a fraction of the object width with the difference between its maximum and minimum  $x$  coordinates).
9. Variance of the width of all instance pieces.
10. Mean rectangularity for the instance pieces.
11. Variance of the rectangularity of all instance pieces.
12. Mean ratio (largest side)/(smallest side) for the instance pieces.
13. Variance of the ratio (largest side)/(smallest side) of all instance pieces.
14. Percentage of large pieces (whose area is greater than  $1/2$  of the object total area).
15. Percentage of small pieces (whose area is less than or equal to  $1/4$  of the object total area).
16. Percentage of right internal angles (respect a to the total angles of all pieces of the instance).
17. Percentage of vertical/horizontal sides (respect a to the total sides of all pieces of the instance).
18. Percentage of high rectangularity pieces (items which rectangularity is greater than 0.9).
19. Percentage of low rectangularity pieces (items which rectangularity is less than or equal to 0.5).

Before linking cluster membership with the problem features, a correlation analysis was performed to detect pairs or sets of features highly related, and thus equivalent. The criteria chosen was an absolute value of correlation index at least of 0.9. Five sets of equivalent features were found:

- Features 4 and 15 (*mean area of instance pieces* and *percentage of small pieces*) had a correlation index of -0.975. *Mean area* is eliminated in order to keep the feature which was actually used in previous work [143].
- *Mean* and *variance of the ratio (largest side)/(smallest side) of all instance pieces* (features 12 and 13) were highly correlated (0.943). We keep *mean* since it is more intuitive than variance.
- Features 10, 17 and 19. A group of three *equivalent* features was found: *mean rectangularity*, *percentage of vertical/horizontal sides* and *percentage of low rectangularity pieces*. Features 10 and 17 had a correlation of 0.968, whether features 10 and 19 were negatively correlated (-0.977) similar than features 17 and 19 (-0.927).
- A group of four *equivalent* features was found: 10, 16, 17 and 18 (*mean rectangularity*, *percentage of right internal angles*, *percentage of vertical/horizontal sides* and *percentage of high rectangularity pieces*). This four size group presented high positive correlation between every possible pair.

Previous group and this one shared features 10 and 17. From these two, we chose to keep *percentage of vertical/horizontal sides* (feature 17). So, features 10, 16, 18 and 19 were taken away.

- Features 2 and 19 (*mean number of sides* and *percentage of pieces with low rectangularity*) has a correlation index of -0.909. *Percentage of pieces with low rectangularity* (feature 19) was already eliminated. Besides, *mean number of sides* may be more intuitive.

Six features were eliminated from the analysis (4, 10, 13, 16, 18 and 19), since they carried what we could call *redundant information*. The remaining 13 features were used to predict instance cluster through MLR<sup>2</sup>. This model predicts correctly the cluster of 80.0% of the instances. This is, by far, greater than 23.67% which is the corresponding by chance (accuracy rate is obtained by squaring and summing the proportion of cases in each group  $(61^2 + 124^2 + 164^2 + 144^2 + 47^2)/541^2$ ). So, the selected features are indeed related with normalized heuristic performance. Nine out of the 13 features used in the model were significantly related with cluster membership ( $p$ -value < 0.05). We selected seven from this set. This features can predict to some extent the way single heuristics behave at instances of the 2D irregular BPP. That is why these features (or their equivalents) are the suggested relevant features to be consider in a representation of the problem state.

Most of these features can be measured in a numerical scale, rather continuous or discrete. Whatever the case may be, practical instances fall in a limited range of values for each characteristic. In this sense, every feature of the problem has an interesting (or practical) range or interval of possible values. The set of instances we count on to develop the analysis must cover this interesting range.

All selected features are not in the same order of magnitude. For example, one feature is measured in a percentage (from 0 to 1), while another feature is measured in thousands of units

---

<sup>2</sup>Multinomial Logistic Regression was performed in the software SPSS for Windows, release 10.0.1 (<http://www.spss.com/>).



and fall in the interval  $(0, \infty)$ . This fact has an implication: the hyper-heuristic search space becomes wider and more complex and the hyper-heuristic would not be scale-invariant as proposed by Ross [127], capable of being applied to problems of very different sizes without extra implementation effort. Some of the hyper-heuristic capability of generalization may be compromised by this change in relevant features. The numerical vector representing the problem state could have elements very different in magnitude perturbing distance calculation. We considered three possible actions to deal with this: (1) to perform a lineal mapping to a fixed scale to all feature values used in the instance state representation. For example, making each possible feature value to fall in the scale from 0 to 1, thus all the numerical vector have values in  $[0, 1]$ ; (2) to use standardized features (with mean 0 and standard deviation of 1). Standardization can be done utilizing mean and variance of every feature for the whole set of experimenting instances. (3) to make the distance from the vector of the problem state representation to the hyper-heuristic blocks a weighted one. Instead of developing a search process for finding weights as in [113], weights could be directly calculated as inversely proportional with features variance or magnitude, since otherwise, most variable features and also larger features would tend to contribute more to the distance calculation. In summary, if some scale correction is not performed, just one or few features would have a significant impact in the distance calculation.

The first scale correction option was chosen. Every feature value was scaled to the interval  $[0, 1]$ . Features that are percentages are naturally in that scale. In every other feature a lineal mapping was done taking as reference the minimum and maximum value of the feature in a set of experimenting instances (see Figure 6.3). Note that not only complete instances are to be represented by the numerical vector, but also every intermediate state. A feature value of an intermediate state could fall in the neighborhood out the range  $[0, 1]$ . That is why the hyper-heuristic model considers labeling points outside this interval.

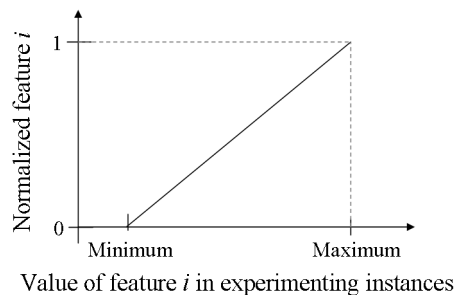


Figure 6.3: Lineal mapping to the interval  $[0, 1]$  for feature measures.

In order to conform the new representation scheme for the instance state, we added an eighth feature regarding the fraction of the instance total items that remain to be packed, so the GA learning process can have a sense of how advanced is the solution of a given instance (this is also done in [143], see Table 6.1). So, the representation scheme for the 2D irregular BPP which is the result of the proposed methodology is shown in Table 6.2. The first seven features in the Table 6.2 are ordered from the most to the less significative according to the MLR model. The number of features selected is the same for representation scheme 1 and 2 for a fair comparison.

Table 6.2: Representation 2 of the instance state.

Feature	Description
1	Number of pieces.
2	Mean number of sides of remaining pieces.
3	Variance of the number of sides of remaining instance pieces.
4	Fraction of remaining pieces in the instance whose area is up to $1/4$ of the object area.
5	Variance of the rectangularity of remaining pieces.
6	Variance of the width of the remaining pieces.
7	Variance of the area of the remaining pieces.
8	Fraction of the instance total items remaining.

Petrovic and Qu [113] found that, with five features in the case representation, the system has the highest performance. Our number of selected features is close to theirs. In Petrovic research, with five features, adjusting the weights of features may not provide much improvement on the system performance, so no weights are needed. Their explanation goes in the direction that there are more complex features, and the similarity measure may have enough information for comparison. Another explanation may be that the features are much more important than their weights in the similarity measure if the features are elaborated enough. Their experimental results showed that the feature selection is more important than their weights. Our hyper-heuristic model does not consider feature weights and Petrovic results about weights [113] says to us, that we are in the right direction.

Analyzing one of the most relevant features, *Mean number of sides of remaining pieces*, related with performance of heuristics BF+BLI ( $h_1$ ) and BFD+CA ( $h_2$ ) (see Table 5.1), we found out that the larger the mean number of sides the better idea to use heuristic  $h_1$  and the worse idea to use heuristic  $h_2$ . This is shown in Figure 6.4. *Absolute* performance for all single heuristics is better as mean number of sides increases (correlation indexes between heuristic performance and mean number of sides go from 0.53 to 0.79 along the 40 heuristics). Mean number of sides goes from 3.2 to 4.1; so, it seems easier to place four-sided pieces than triangles. Nevertheless, heuristic  $h_2$  decreases its *normalized* performance as mean number of sides increases. This means that, heuristic  $h_2$ , although increases its absolute performance while mean number of sides increases (like all other heuristics), this performance does not improve as much as other heuristics do, making heuristic  $h_2$ , in general, less adequate for solving instances with higher mean number of sides.

The opposite happens to the same two heuristic comparing their normalized performance with the feature *variance of the number of sides* (see Figure 6.5). In this case, it is the normalized performance of heuristic  $h_1$  the one that decreases, while heuristic  $h_2$  improves its relative performance (relative against the other heuristics) as the variance of the number sides increases. That is,  $h_2$  is, in general, more adequate for solving instances with higher variance of the number of sides.

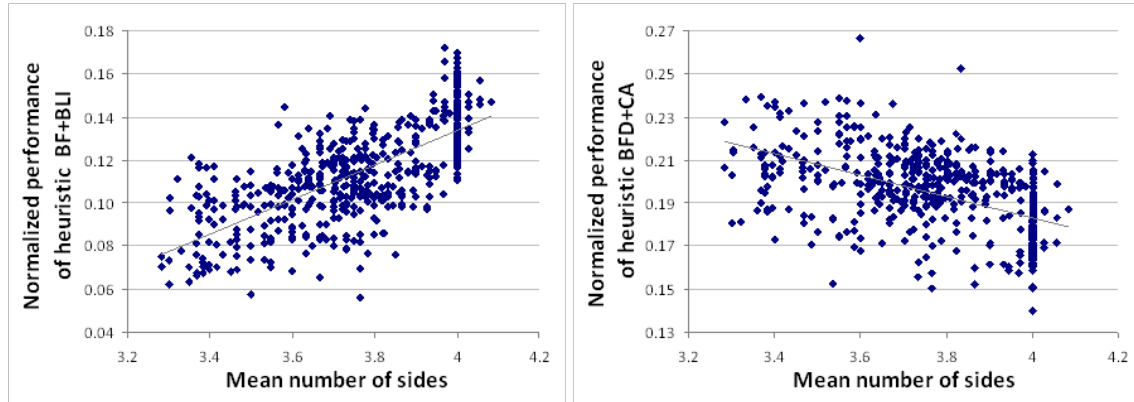


Figure 6.4: Normalized performance of heuristics BF+BLI ( $h_1$ ) and BFD+CA ( $h_2$ ) related with instances mean number of sides.

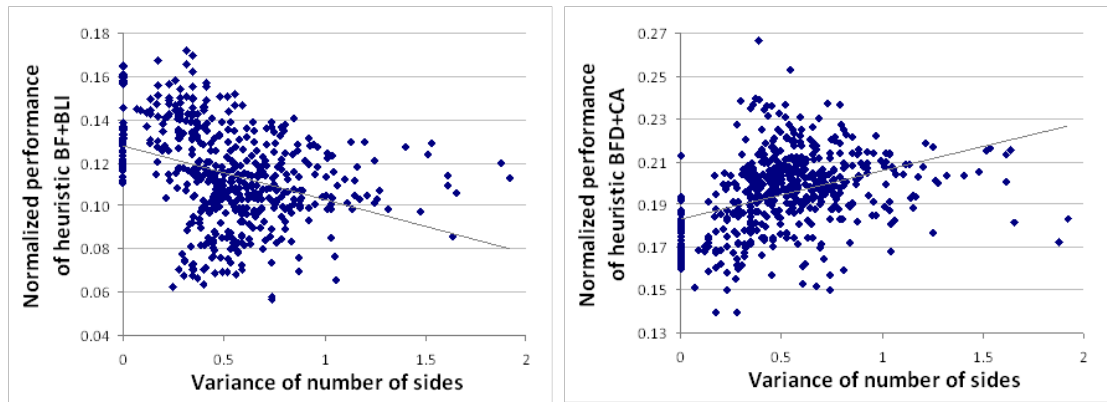


Figure 6.5: Normalized performance of heuristics BF+BLI ( $h_1$ ) and BFD+CA ( $h_2$ ) related with the instance variance of the number of sides.

This is what can be said when analyzing this feature alone. It is expected some interaction effect among several feature values, and this justifies the use of an evolutionary computation based approach for developing a set of rules for heuristic selection.

## 6.3 Experiments

This section compares the quality of the hyper-heuristics generated when instances states are summarized employing representation schemes 1 and 2 (Tables 6.1 and 6.2). The first one is developed based on a series of parameters related to the problem domain and selected by intuition. The second one uses data mining techniques to determine the feature set.

The problem instances considered are described in Section 4.2.2. As well as in our first model implementation (Chapter 5), we also added a problem from the literature named *Fu* [54]. We have a total of 541 irregular problem instances. Instance characteristics are listed in Tables 4.1 and 4.3. Number of pieces per instance are in the range from 12 to 60.

In this investigation ten selection heuristics and four different placement heuristics were

employed. All possible combinations of selection and placement heuristics conform the *single heuristic repository*  $H$ , that is the set of possible actions available to solve an instance (see Table 5.1). Here,  $|H| = 40$ . Chromosome representation is the same as in Chapter 5 and the fitness function employed to measure chromosome quality is computed as explained in Section 3.4. The available problem instances are divided into a training and a testing set. The GA is used with the training set only, until a termination criterion is met and a general hyper-heuristic has been evolved. All instances in both sets are then solved with this hyper-heuristic.

Two experiments were conducted sorting the 541 available instances in two balanced training and testing sets and other two experiments swapped training and testing sets (full description in Section 5.2). Four experiments overall. For each of the four experiments with each representation scheme (1 and 2), two GA processes were run. For each complete run the best two individuals of all over the process were selected as hyper-heuristics. That is, four hyper-heuristics were obtained for every representation scheme in each of the four experiments. Overall, sixteen hyper-heuristics were employed to measure the effectiveness of each representation scheme.

We changed some GA parameters with respect previous work with the hyper-heuristic model (presented in Section 5.2). After some preliminary experimentation, population size was reduced from 100 to 30. Since the GA is steady-state and two members of the population are changed at a time, the reduction of the population size helps the population to evolve faster. Crossover probability was kept in 1.0. Preliminary experiments showed that a mutation probability of 0.1 produced better results than the previous probability of 0.05. The diversity of the search space was explored by doing two replicas for each of the four experiments and increasing the mutation rate. So now, in 80 generations we have good chances of finding a high rated individual in contrast with 500 generations used in [143]. These new parameters represent a huge reduction of the number of evaluations of the fitness function to one-twentieth. Experiments with both representation schemes were performed with these new parameters.

We chose the chromosome with highest fitness of all over the process as the hyper-heuristic we seek. In [143], the highest fitness chromosome of the very last generation had been chosen, based on the idea that the final generation is the best generation and the one with more accurate fitness chromosomes since they are older (evaluated more times). We want a chromosome to be good at solving many problems. When a chromosome is created, it is applied to just five randomly-chosen problems in order to obtain an initial fitness. Then, at each cycle of the algorithm, a new problem is given to each of the chromosomes to solve. This procedure is a trade-off between accuracy and speed [128]. The longer a chromosome survives in a population the more accurate its fitness becomes, which raises the question of how to rate a young chromosome (with therefore a poor estimate) against an older chromosome (with an improved estimate) if the fitness of the younger one is better?. In [128] several methods were tried to attempt to give higher credit to older and therefore more accurate chromosomes; however, these trials rather surprisingly indicated that age of chromosome was not an important factor. This may be due to the fact that the fitness of every chromosome computed through the GA process is an unbiased estimator of its true fitness (true fitness is the one that would had been computed solving all instances of training set), because every individual is evaluated with a random sample of instances. This means that if a young individual is better rated than an older one; actually it is more probable that the former is better than the latter when solving

Table 6.3: Number of extra objects delivered by hyper-heuristics using representation scheme 2 compared with those using representation scheme 1. Experiments I through IV.

	Training Sets				Testing Sets			
	I	II	III	IV	I	II	III	IV
$\leq -4$		10.0						
-3	0.4	1.1	0.4	2.6			1.1	1.5
-2	3.3	1.1	7.0	3.0	0.4	1.1	8.1	3.0
-1	14.0	45.6	40.2	11.9	11.9	25.8	38.5	10.7
0	72.3	41.5	52.0	80.4	74.4	66.1	51.5	81.9
1	10.0	0.7	0.4	2.2	13.0	6.3	0.7	3.0
2					0.4	0.7		

a great number of instance problems; even though the distance between the estimated fitness and the true fitness is likely to be greater in a younger chromosome. The population average fitness tends to be higher generation over generation; but the best individual fitness of each generation, which also has an upward tendency, behave in a more random fashion. It is not unusual to find a very high rated individual in earlier generations. That is why, we discarded any attempt to make any special concessions to elder chromosomes in present research.

## 6.4 Results and Discussion

Each hyper-heuristic generated was used to solve the training and testing sets of instances of the experiment where it comes from. Results are shown in Table 6.3. Figures in cells indicate the percentage of problems solved by hyper-heuristics with representation 2 that employs a particular number of extra objects (left column) when compared against results provided by the previous representation scheme. For example, when solving the training set instances of Experiment II, hyper-heuristics developed using representation scheme 2 required at least 4 objects less in 10.0% of instances when comparing with hyper-heuristics developed using representation scheme 1. In previous experimentations, it has been confirmed that solving an instance with a hyper-heuristic is faster than solving it with each of the single heuristics and then choosing the best result, as well as that the method to create hyper-heuristics and the hyper-heuristics themselves are efficient with respect to the number of objects used [143].

Table 6.4 summarizes the results for all experiments. Comparing the average number of objects employed, the proposed new representation scheme uses 0.446 and 0.217 fewer objects for the training and testing sets respectively, averaging over results of the 16 hyper-heuristics. In Table 6.4 most numbers are negative, meaning that representation 2 deliver in average fewer objects than representation 1. We found out that the number of objects employed by hyper-heuristics is statistically lower with representation scheme 2 than with representation scheme 1 ( $p$ -value  $< 0.001$ ). The sample size was 541 instances.

Figure 6.6 shows the average fitness of the whole population for the eight GA processes run for each representation scheme. Although at the initial generations average fitness tend to

Table 6.4: Average number of extra objects obtained by hyper-heuristics using representation scheme 2 compared with those using representation scheme 1.

Exp.	Training	Testing
I	-0.104	0.037
II	-0.959	-0.203
III	-0.525	-0.534
IV	-0.196	-0.166
Avrg	-0.446	-0.217

be equal, representation scheme 2 generates better average individuals, in later generations.

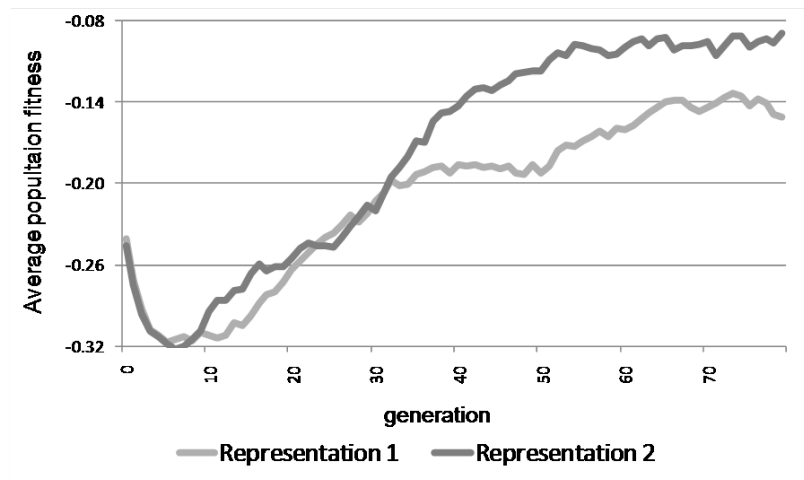


Figure 6.6: Average fitness of the whole population for all experiments conducted under each representation scheme.

For each instance we computed the average number objects employed by the 16 hyper-heuristics generated for each representation scheme and then subtract the number of objects employed by the instance’s best single heuristic. Then, we found the difference from the results of representation 1 against those from representation 2 and compare these differences to zero performing a t-test (statistic  $T = -15.07$ ,  $p$ -value  $< 0.001$ ). The sample size was 541. We concluded that the number of extra objects (compared with the best single heuristic for each instance) employed by hyper-heuristics is statistically lower with representation scheme 2 than with representation scheme 1.

## 6.5 Summary

We have proposed a way to define a numerical vector representation for an instance of a given problem. The idea is to find a set of features able to predict to some extent heuristic performance for a training set of problem instances. Applying this methodology to the 2D

irregular BPP we found a group of features to conform the representation scheme. We showed that building hyper-heuristics with the proposed representation scheme produces considerable better results than employing the representation scheme used in [143] (Chapter 5). The developed methodology could be applied to any optimization approach that needs to represent or summarize instances states through numerical vectors. For example, the Case-based reasoning approach [113] or hyper-heuristics approaches where a number of features is used to categorize instances for choosing an adequate heuristic [128, 129, 138, 139, 141].

Next chapter studies deeply the DJD heuristic (formerly introduced in Section 2.6.1) and implements a time-saving routine when it is used for solving 2D instances.





# Chapter 7

## An Effective Heuristic for the 2D Irregular BPP

This chapter proposes an adaptation, to the two-dimensional irregular bin packing problem of the Djang and Finch heuristic (DJD), originally designed for the one-dimensional bin packing problem. This heuristic selects the next pieces to be placed. In the two-dimensional case, not only is it the case that the piece's size is important but its shape also has a significant influence. Therefore, DJD as a selection heuristic has to be paired with a placement heuristic to completely construct a solution to the underlying packing problem. We found a simple but successful placement procedure that produces outstanding results when coupled with the proposed selection heuristic (the Constructive Approach (Maximum Adjacency) (CAD) from section 2.6.2). A successful adaptation of the DJD requires a routine to reduce computational costs, which is proposed and successfully tested in this dissertation. Results, on a wide variety of instance types with convex polygons, are found to be significantly better than those produced by more conventional selection heuristics. The proposed heuristic is not only fast in execution, but it also produces excellent results.

For testing the procedure, we employed randomly generated instances. As stated in section 1.1, as far as we know, there are no previous studies for, specifically, the 2D irregular single bin size bin packing problem. Therefore, there are not 2D irregular BPP instances available in the literature. All 2D irregular problem instances available are intended for the strip packing problem. Although problems statements for the 2D irregular BPP and the strip packing problem are similar, results are not comparable. This means that this dissertation and derived papers present the first results regarding the performance of single-pass constructive heuristics for the 2D irregular BPP. Nevertheless the brief research in the 2D irregular BPP, there exist many practical applications where irregular pieces are cut from identical rectangular objects (such as in shipbuilding industry where free-form shapes are cut from rectangular steel plates [109]).

The chapter proceeds as follows. The next section describes the DJD heuristic, which is the building block for our approach. Section 7.2 describes the implementation details of the heuristic. Sections 7.3 and 7.4 give the experimental details and results respectively. Finally, section 7.5 presents a summary of the chapter.

## 7.1 The DJD Heuristic

The proposed approach is based on the DJD heuristic, which is a selection heuristic designed for the 1D case. In its original version, as explained by [129], the DJD heuristic puts items into a bin, taking items largest-first until that bin is at least one third full. It then tries to find one, or two, or three items that completely fill the bin. If there is no such combination it tries again, but looking instead for a combination that fills the bin to within 1 unit of its capacity. If that fails, it tries to find a combination that fills the bin to within 2 units of its capacity; and so on. This process is outlined in the pseudo code of the Algorithm 8. This routine is to be performed as long as there are pieces to place. DJD is a single-pass constructive heuristic. A popular variation of DJD is called DJT (Djang and Finch, more tuples) which considers combinations of up to five items rather than three items.

---

**Algorithm 8** The original DJD heuristic.

---

**Require:** A list of pieces in descending order; size of objects.

**Ensure:** All pieces placed in objects.

$waste = 0$ ;  $w = 1$  [increment of allowed waste]

**while** there are pieces to place **do**

    Fill the last open object until is at least one third full

    Register in memory every piece that does not fit

    Try pieces one by one

**if** a piece could be placed **then**

        reset  $waste = 0$  and start again trying pieces one by one

    Try groups of 2 pieces

**if** a pair of pieces could be placed **then**

        reset  $waste = 0$  and start again trying pieces one by one

    Try groups of 3 pieces

**if** a group of 3 pieces could be placed **then**

        reset  $waste = 0$  and start again trying pieces one by one

**if** no piece could be placed trying all possible 1, 2 or 3-piece groups

    AND  $waste < \text{object free area}$  **then**

$waste = waste + w$

**else**

        open a new object

---

In [104, 128, 129], DJD and its variation DJT have been implemented for the 1D BPP as a part of procedures (hyper-heuristics) that learn to combine heuristics for solving the underlying problem. In these approaches, the idea is to automatically apply different heuristics to different states of the construction process. In this scenario, DJD and DJT were reported as the best heuristics considered. Also, the DJD heuristic was adapted to solve the problem of

scheduling transportation events for minimizing the number of vehicles used, while satisfying the customer demand [144]. Kos and Duhovnik [91] describe the same heuristic but named it as *Exact Fit* in an approach for rod cutting optimization with remnants utilization.

Although the DJD heuristic, exactly as stated, works well in many problems known to be hard, it fails in other types of problem. For example, consider a very easy problem in which the bins have capacity 1000 and there are 10,000 items each of weight 1. Packing these items will need only 10 bins. However, DJD will first fill a bin until it contains 334 items (just over one-third) and then add just three more items into the bin, so the bin will contain 337 items. Thus, 30 bins will be needed ( $337 \times 29 = 9773$ ) [126], a solution far from optimality. The obvious remedy to this situation is to keep trying to place item combinations until no single item can be placed. Although, in the case when items are so small compared with the bin free space, there is no advantage in trying every combination of items, since no combination would result in zero waste.

For the 2D regular case, where pieces to be placed are rectangles, DJD and DJT have been adapted and implemented as a selection heuristic [138, 139]. However, the authors did not report, in this case, a routine for improving the running times. This is essential in the 2D case, because simply comparing the area of a 1, 2 or 3-piece combination against the free area of the object does not imply that the pieces can actually be placed. Indeed, several groups of pieces may need to be tried before a given combination of pieces can be placed. Moreover, the same pieces may be tested several times in different combinations before the algorithm is successful in placing a 1, 2 or 3-piece group. Besides, to determine whether or not a piece can be placed in a given object is the most time-consuming task when solving a 2D Bin Packing Problem. The placement task requires even more running time when pieces are irregular. For the 2D irregular BPP, where pieces to be placed are convex polygons, DJD has been implemented as a member of a heuristic repository in a hyper-heuristic approach [143]; but the performance of DJD was not analyzed, nor reported separately. Furthermore, the authors did not report a routine for improving the running times. To our knowledge, DJT has not been implemented for the 2D irregular BPP. In this chapter, DJD is adapted to and thoroughly analyzed when solving a variety of instances of the 2D irregular BPP. Moreover, a routine for reducing redundant computation is proposed and successfully tested.

## 7.2 The Proposed DJD Heuristic for the 2D Irregular BPP

The DJD algorithm for the 2D case works as a selection heuristic, but it alone does not solve the problem completely. DJD has to be paired with a placement heuristic which will determine the exact position of each piece inside an object.

In the 2D adaptation of the heuristic, DJD puts pieces into an object, taking them by decreasing size, until that object is at least one third full. It then tries to find one, or two, or three pieces that completely fill the object. If there is no such combination it tries again, but looks instead for a combination that fills the bin to within  $w$  of its capacity. If that fails, it tries to find such a combination that fills the object to within  $2w$  of its capacity; and so on. In the 1D case, the waste incremental suggested is 1 unit. Depending on the order of magnitude of the object and pieces sizes, in 2D it would not be feasible to manage a 1-unit incremental. Therefore, the incremental should be selected according to the total object area. For the 2D

adaptation of the heuristic, the processes of reviewing groups of one, two or three pieces are modified to optimize running time. These processes mentioned in Algorithm 8 are described in Algorithms 9, 10 and 11 respectively.

---

**Algorithm 9** The proposed DJD algorithm. Trying pieces one by one.

---

```

for all pieces in decreasing order of size do
  if object free size – size of piece > waste then
    break
  if size of piece > object free size OR piece has failed to fit then
    continue
  Try to place the piece in last open object
  if the piece could be placed then
    return
  else
    register in memory that the piece does not fit

```

---



---

**Algorithm 10** The proposed DJD algorithm. Trying groups of 2 pieces.

---

```

for all pieces in decreasing order of size do
  if object free size – size of piece – size of greatest piece > waste then
    break
  if the piece has failed to fit OR piece's size + smallest piece's size > free space then
    continue
  Try to place the piece in last open object
  if the piece could not be placed then
    register it in memory
  else {select a second piece}
    for all possible second pieces in decreasing order of size do
      if object free size – size of the 2 pieces > waste then
        break
      if the pair of pieces or any piece has failed to fit OR 2 pieces' size > free space then
        continue
      Try to place the second piece in last open object
      if the piece could be placed then
        return
      else
        unplace first piece AND register that the pair of pieces does not fit

```

---

Every time a combination of 1, 2 and 3 pieces is placed, the checking process starts all over again in the same object (resetting the allowed waste to 0). When no more pieces can be placed in an object, a new object is opened. The DJD heuristic works in one open object at a time, there is no need to review previous opened objects. Order is important in 2D packing, groups with the same pieces are revised considering all possible orderings. A piece combination that cannot be placed in a particular order could be placed in another piece order.

---

**Algorithm 11** The proposed DJD algorithm. Trying groups of 3 pieces.

---

```

for all pieces in decreasing order of size do
  if object free size – size of piece – size of the 2 greatest pieces > waste then
    break
  if the piece has failed to fit OR piece's size + 2 smallest pieces' size > free space then
    continue
  Try to place the piece in last open object
  if the piece could not be placed then
    register it in memory
  else {select a second piece}
    for all possible second pieces in decreasing order of size do
      if object free size – size of the 2 pieces – size of greatest piece > waste then
        break
      if the piece or the pair of pieces has failed to fit OR
      size of the 2 pieces + size of smallest piece > object free size then
        continue
      Try to place the second piece in last open object
      if the piece could not be placed then
        unplace first piece AND register that the pair of pieces does not fit
      else {select a third piece}
        for all possible third pieces in decreasing order of size do
          if object free size – size of the 3 pieces > waste then
            break
          if any piece, or pair or 3-piece group of pieces have failed to fit OR
          size of the 3 pieces > object free size then
            continue
          Try to place the third piece in last open object
          if the piece could be placed then
            return
          else
            unplace first 2 pieces AND register that the 3-piece group does not fit

```

---

As it can be seen in Algorithms 9, 10 and 11, when DJD checks one, two or three-piece groups, first it compares the pieces' areas against the maximum waste allowed and against the available object area. Only then, does DJD try to place them. For the 2D BPP, checking if a piece could or could not be placed is computationally expensive. Pieces should be in descending order when the DJD heuristic starts, allowing the *For* cycles to *break* at some point when reviewing pieces; thus, reducing many comparisons (see Algorithms 9, 10 and 11).

In order to reduce the computational effort, for each object, a record of what pieces have been tried so far as a first member of a 1, 2 or 3-pieces group is kept, so the algorithm does not try again the same piece in a different piece group. Additionally, a record is kept of all ordered pairs of pieces that failed to be placed in a particular object either as a 2-piece group or as the first 2 pieces of a 3-piece group. These pairs of pieces are, therefore, not tried again in the same order. Finally, all ordered 3-piece groups that fail to fit in an object are recorded as well. These records help to reduce an important amount of redundant computation.

According to the placement procedures considered, when a piece cannot be placed in an object at a given time, there is a slight possibility that it can actually be placed later when one or more pieces had been placed. Considering this possibility in the implementation would increase the algorithm running time. If time is not a constraint, the option would be to keep a record of pieces that fail to fit just until one piece or group is placed, and then clean up the records.

## 7.3 Experiments

This section describes how the DJD is tested against seven other selection heuristics combined with four different placement heuristics.

We employ the 540 problem instances described in Section 4.2.2 which are 2D irregular BPP (convex), except the *Fu* instance since it was always solved with 2 objects by every single heuristic. The quality measure is computed as in equation 3.2.

### 7.3.1 The Other Selection Heuristics and the Placement Heuristics

The selection heuristics used for comparison against DJD are:

1. First Fit (FF)
2. First Fit Decreasing (FFD)
3. First Fit Increasing (FFI)
4. Filler + FFD
5. Best Fit (BF)
6. Best Fit Decreasing (BFD)
7. Worst Fit (WF)

These seven heuristics are all the single-pass selection heuristics that we could get from the literature for the offline BPP, in which the list of pieces to be packed is static and given in advance. Description of each heuristic is found in Section 2.6.1. Notice that the first part of DJD, when an object is filled until one-third, corresponds to the FFD heuristic.

Once a piece and an object are selected, the placement heuristic states the way in which the piece is located inside the object. Two different placement heuristics could arrive to different conclusions as to whether a piece can or cannot be placed inside the object, and about the piece's final coordinates. We consider four placement heuristics that work in combination with the selection heuristics:

1. Bottom-Left (BL)
2. Constructive Approach (CA)
3. Constructive Approach (Minimum Area) (CAA)
4. Constructive Approach (Maximum Adjacency) (CAD)

Explanation of each placement heuristic is found in section 2.6.2. For the last listed three placement heuristics: CA, CAA and CAD, the algorithm rotates each piece by multiples of 90 degrees and chooses the rotation that is better according to each heuristic criterion. For the first heuristic, BL, no rotation is considered, since BL does not choose among several possible positions as the other three placement heuristics do.

Our empirical study explored all combinations of selection and placement heuristics with each of the available instances.

## 7.4 Results and Discussion

The value of the waste incremental  $w$  is an important choice in the DJD heuristic. As observed experimentally in our instance set, if the waste incremental is set to  $w = 1$ , many 1-unit increments occur during the solution construction at a high computational cost without placing any single piece. We empirically found that a waste incremental of one-twentieth of the total object area is a good balance between fast and good solutions. Increments lower than  $w = 1/20$  of the object total size have a high computational cost, without a significant improvement in fitness, while increments higher than  $w = 1/20$  of the object size lead to inferior results.

We explored the phenomenon of a piece that cannot fit into an object and it later fits (when there are one or more pieces in the object). This is rare for placement heuristics CA, CAA and CAD. Therefore, in this case, trying to fit pieces after they have failed to be placed, increases the running time. Results may be slightly better (or worse) in some cases, but generally speaking, the small improvement does not pay the huge excess of processing time (although this would depend on the particular application). For the BL placement heuristic this phenomenon is less rare. Hence, when using BL in combination with any selection heuristics, a record of pieces that fail to be placed is kept until a piece is successfully placed. After that, the records are cleaned.

We explored different initial levels of fullness before trying to place combinations of pieces within an allowed waste, namely,  $1/4$ ,  $1/3$ ,  $1/2$  and  $2/3$ . DJD heuristics with these

levels are referred to as  $DJD_{1/4}$ ,  $DJD_{1/3}$ ,  $DJD_{1/2}$  and  $DJD_{2/3}$ , respectively. Along with the 7 selection heuristics described above, we have 11 different selection heuristics overall.

All instances were solved with all heuristics (11 selection heuristics  $\times$  4 placement heuristics = 44 ways to solve a given instance). Figure 7.1 shows the solution of an instance type  $C$  with the selection heuristic  $DJD_{1/3}$  and the placement heuristic CAD.

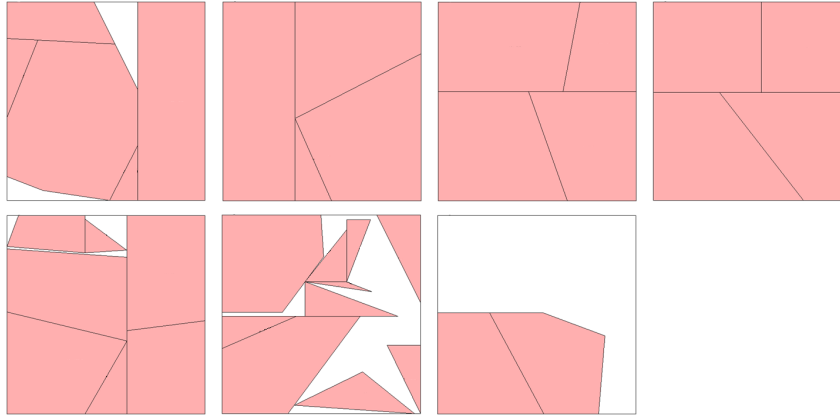


Figure 7.1:  $DJD_{1/3}$  in combination with CAD heuristic solves an instance of type  $C$ . This solution needs one more object than the optimal solution in which there would be zero waste. In this solution, fitness is 0.776 measured with equation 3.2

Table 7.1 shows the average fitness for every possible combination of selection and placement heuristics along the 540 instances. Two variants of the DJD heuristic,  $DJD_{1/3}$  and  $DJD_{1/4}$ , outperformed the other selection heuristics tried. The best combination of selection and placement heuristic is  $DJD_{1/3} + CAD$ , closely followed by  $DJD_{1/4} + CAD$ . The placement heuristic CAD is clearly the best no matter which selection heuristic it is paired with. For the different variations of the DJD tried,  $DJD_{1/4}$  is the best when used along with the BL and CA placement heuristics and  $DJD_{1/3}$  is the best when used along with the CA, CAA and CAD placement heuristics. Therefore, we found that the one-third of the object capacity for the initial fullness before trying different combinations of pieces, as stated by the original version of the DJD for 1D BPP, is also suitable for the 2D irregular BPP.

Table 7.1: Average fitness for all the combinations of selection and placement heuristics over the 540 instances.

Placement Heuristics	Selection Heuristics											Average
	FF	FFD	FFI	Filler	BF	BFD	WF	DJD 1/4	DJD 1/3	DJD 1/2	DJD 2/3	
BL	0.347	0.422	0.302	0.426	0.348	0.423	0.306	<b>0.485</b>	0.472	0.435	0.429	0.400
CA	0.439	0.563	0.352	0.569	0.437	0.564	0.385	<b>0.583</b>	<b>0.583</b>	0.566	0.563	0.510
CAA	0.436	0.560	0.350	0.567	0.438	0.562	0.373	0.574	<b>0.576</b>	0.561	0.562	0.505
CAD	0.501	0.648	0.383	0.650	0.501	0.650	0.421	0.682	<b>0.683</b>	0.653	0.649	0.584
Average	0.431	0.548	0.347	0.553	0.431	0.550	0.371	<b>0.581</b>	0.578	0.554	0.551	0.500

For the problem instances from  $A$  to  $R$ , the CAD heuristic produced better average performance when combined with all the selection heuristics. Table 7.2 shows the average fitness obtained by all the selection heuristics with the CAD placement heuristic. Results are reported



Table 7.2: Average fitness obtained by all the selection heuristics when combined with the CAD placement heuristic for each instance type. The best selection heuristic for each instance type is in bold font.

Instance Type	Selection Heuristics										
	FF	FFD	FFI	Filler	BF	BFD	WF	DJD <sub>1/4</sub>	DJD <sub>1/3</sub>	DJD <sub>1/2</sub>	DJD <sub>2/3</sub>
A	0.486	0.600	0.371	0.601	0.491	0.600	0.379	0.598	0.596	<b>0.605</b>	0.599
B	0.606	0.753	0.460	0.753	0.611	0.754	0.549	<b>0.929</b>	<b>0.929</b>	0.756	0.753
C	0.515	0.704	0.381	0.701	0.506	0.709	0.421	0.751	<b>0.763</b>	0.723	0.702
D	0.411	0.578	0.338	0.576	0.410	<b>0.579</b>	0.362	0.574	0.566	0.576	0.573
E	0.301	<b>0.412</b>	0.230	0.411	0.298	0.411	0.224	0.393	0.399	0.403	0.406
F	0.393	0.493	0.279	<b>0.496</b>	0.388	0.493	0.297	0.491	0.493	0.493	0.494
G	0.592	0.707	0.448	0.707	0.601	0.708	0.520	<b>0.814</b>	<b>0.814</b>	0.708	0.707
H	0.603	0.747	0.458	0.746	0.622	0.747	0.518	<b>0.928</b>	<b>0.928</b>	0.746	0.746
I	0.598	0.619	0.573	0.624	0.598	0.619	0.577	0.621	0.619	0.626	<b>0.627</b>
J	0.543	0.661	0.457	0.664	0.538	0.662	0.462	0.652	0.659	0.660	<b>0.665</b>
K	0.541	0.709	0.385	0.713	0.526	0.710	0.447	0.706	<b>0.718</b>	0.700	0.708
L	0.349	0.485	0.278	0.494	0.347	0.485	0.290	<b>0.512</b>	0.499	0.502	0.489
M	0.417	0.579	0.307	0.580	0.406	0.582	0.337	0.573	<b>0.589</b>	0.584	0.58
N	0.446	0.495	0.290	<b>0.503</b>	0.445	0.495	0.371	0.493	0.493	0.497	0.499
O	0.537	0.791	0.409	0.791	0.545	0.787	0.432	<b>0.823</b>	0.812	0.791	0.81
P	0.481	0.661	0.358	0.664	0.483	0.662	0.396	<b>0.678</b>	<b>0.678</b>	0.663	0.657
Q	0.672	0.942	0.483	0.943	0.670	0.972	0.558	0.967	0.977	<b>1</b>	0.946
R	0.533	0.724	0.390	0.726	0.538	0.726	0.442	<b>0.771</b>	0.753	0.725	0.723
Average	0.501	0.648	0.383	0.650	0.501	0.650	0.421	0.682	<b>0.683</b>	0.653	0.649

Table 7.3: Average computational time (in seconds) for all the combinations of selection and placement heuristics over the 540 instances.

Placement Heuristics	Selection Heuristics											Average
	WF	BF	FF	FFI	FFD	BFD	Filler	DJD <sub>1/4</sub>	DJD <sub>1/3</sub>	DJD <sub>1/2</sub>	DJD <sub>2/3</sub>	
BL	0.01	0.02	0.01	0.02	0.02	0.02	0.04	1.12	0.93	0.68	0.32	0.29
CA	0.41	0.76	0.83	0.85	1.20	1.18	5.77	10.04	9.72	7.83	6.61	4.11
CAA	0.44	0.81	0.80	0.86	1.12	1.22	5.45	9.89	9.54	7.89	6.62	4.06
CAD	0.47	0.95	0.94	0.90	1.33	1.42	6.18	12.52	12.37	10.4	9.02	5.14
Average	0.33	0.63	0.64	0.66	0.92	0.96	4.36	8.40	8.14	6.70	5.64	3.40

for the CAD placement heuristic only as it produced the best performance. Type *I* and *J* are the only instance types where  $DJD_{2/3}$  outperformed all others, and type *I* instances are the only ones where all pieces are regular (rectangles). Apart from type *I*, type *J* instances have the highest percentage of right angles. It seems that  $DJD_{2/3}$  goes well along with rectangles. All type *Q* instances are solved to optimally by  $DJD_{1/2}$  using CAD placement heuristic, and 73% of *Q* instances were solved to optimally by  $DJD_{1/3}$  + CAD. Optimum solutions of all type *Q* instances have exactly four pieces in each of 15 objects. Several instances of types *B*, *H* and *O* are also solved to optimally by several variations of DJD.

Computation times, measured in seconds, are shown in Table 7.3. The first seven selection heuristics are listed in Table 7.3 from fastest to slowest (left to right). We ran the algorithms on an Intel Core Duo 2.33 GHz PC. All the combinations of selection and placement heuristics produced results within reasonable time (below 10 seconds). However, the DJD variants have longer running times. It was observed experimentally, for  $DJD_{1/3}$  + CAD, that there is an average computational time reduction of 80% when compared to the case where no record is kept at all.

In order to assess the statistical significance of the results, we conducted the hypothesis testing procedure called one-way repeated measures, ANOVA. We employed the solution of each available instance with the eleven different selection heuristics (including the four variations of the DJD heuristic) in combination with the CAD placement heuristic. Considering a sample size of 540 instances, we rejected the hypothesis that observed differences were due to chance in at least a pair of heuristics ( $p$ -value  $< 0.0001$ , the Greenhouse-Geisser correction was used because the assumption of sphericity has not been met). So, at least one heuristic is significantly different from the others. In order to determine which heuristics may be considered to be of similar performance and which not, we performed multiple pairwise comparisons with the Bonferroni adjustment and a significance level of 0.05. Figure 7.2 ranks the eleven selection heuristics considered and connects those that perform equivalently according to the Bonferroni procedure. The FFI heuristic produced the overall lowest fitness, and it is significantly different from the other heuristics. From lowest to highest fitness, the next heuristic is WF; then, BF and FF form a group of similar heuristics since their fitness is not significantly different. Two variations of the DJD heuristic ( $DJD_{2/3}$  and  $DJD_{1/2}$ ) along with FFD, Filler and BFD perform in a similar way along the set of considered instances. The heuristics  $DJD_{1/4}$  and  $DJD_{1/3}$  are the best, and are significantly different from the rest. Although  $DJD_{1/3}$  performs slightly better than  $DJD_{1/4}$ , this difference is not statistically significant.

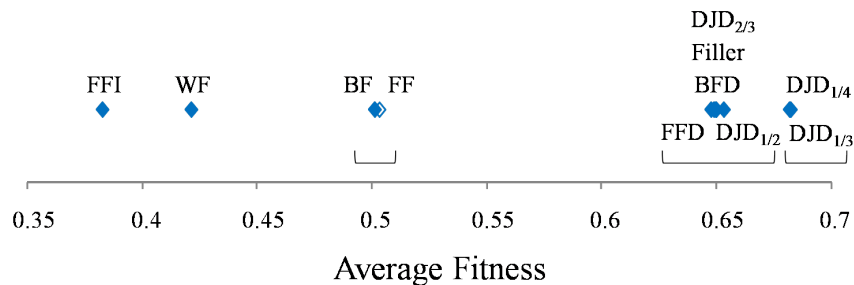


Figure 7.2: Comparison of means for the 11 heuristics considered, using the Bonferroni adjustment.  $DJD_{1/4}$  and  $DJD_{1/3}$  are the better heuristics and there is not significant difference between them

## 7.5 Summary

This chapter of the present dissertation proposed an adaptation, to the 2D irregular Bin Packing Problem, of the Djang and Finch heuristic originally designed for the one-dimensional bin packing problem. Four variants of the DJD heuristic (with initial fullness of  $1/4$ ,  $1/3$ ,  $1/2$  and  $2/3$ , before combinations of pieces are tried to be placed within an allowed waste) were explored and compared with several alternative selection heuristics in the literature. Several placement heuristics were explored and the Constructive Approach with Maximum Adjacency (CAD) was found to outperform the others in this study. Also, the value of the waste incremental is an important choice in the DJD heuristic. We found, empirically, that a waste incremental of one-twentieth of the total object area represents a good balance between fast and good solutions.

An empirical study was conducted over 540 irregular convex instances of different types and a wide range of characteristics. The proposed DJD heuristic was found to statistically outperform the alternative selection heuristics. Moreover, the computational time, although longer for the DJD variants is still within reasonable bounds, which was achieved by a routine keeping appropriate records to reduce the amount of redundant computation. The extra time employed may be due to the extra number of attempts to place pieces. The DJD variants with  $1/4$  and  $1/3$  initial fullness levels were the best performing. Therefore, the one-third of the object capacity for the initial fullness before trying different combinations of pieces, as stated by the original version of the DJD for the one-dimensional case, is also suitable in two dimensions.

The proposed DJD heuristic can be considered as a midpoint between the other very fast single-pass heuristics reviewed in this chapter and other even more complex approaches, like the proposed by Bennell and Song, in which many parallel partial solutions can be generated and compared [14].

In next chapter, a solution approach based on hyper-heuristics is proposed for the 1D BPP and several kinds of 2D BPP (including concave polygons).



# Chapter 8

## Hyper-heuristics for 1D and 2D Bin Packing Problems

This chapter proposes an evolutionary hyper-heuristic framework that is useful for solving 1D and 2D BPP. In the past, these problems had been tackled separately by using a variety of methods and techniques because they exhibit different properties. The proposed model was applied to one-dimensional BPP [128], and later adapted separately to the 2D regular [138] and 2D irregular (convex) packing problems [143]. Now, we integrate these problems into only one framework and extend the hyper-heuristic solution model to instances including non-convex polygons increasing the level of geometrical complexity and computational burden. Our new framework produces hyper-heuristics that can solve a wide range of 1D and 2D BPP instances with good results and without any additional adjustment to solve either 1D instances or 2D irregular instances. With our framework, once a hyper-heuristic has been evolved using a training set of instances, it can be reused on any new instance of any kind producing competitive results, and comparable against those provided by the best single heuristic per problem. We found some interesting patterns about heuristic alternation when solving instances with hyper-heuristics.

In this chapter, we consider the following three problems in Wäscher, Haussner and Schumann's typology [154]:

1. The 1D single bin size bin packing problem,
2. The 2D regular single bin size bin packing problem
3. The 2D irregular single bin size bin packing problem.

In the 1D BPP, there is an unlimited supply of bins, each with capacity  $c > 0$ . A set of  $n$  items (each one of size  $s_i < c$ ) is to be packed into the bins, the task is to minimize the total number of bins used. The 2D BPP is fully described in the Problem Statement Section of this dissertation (Section 1.1). It is common that heuristic approaches for the 2D Bin Packing Problem present at least two phases: first, the *selection* of the next piece to be placed and the corresponding object to place it; and second, the actual *placement* of the selected piece in a position inside the object according to a given criteria. Some approaches consider a third phase as a local search mechanism. For the 1D BPP, the second phase (placement procedure) is not necessary.

Section 8.1 refers to some adjustments made to our solution model to suit our wider range of problems. This section also includes the process of selecting relevant features for the representation scheme. Then, Section 8.2 explains all details regarding the experimentation process. Later, Section 8.3 presents main results showing how single heuristics behave in this solution model.

## 8.1 Implementation

The solution model is presented in detail in Chapter 3. In this section some particular issues of the implementation are discussed.

**Set of Heuristics.**- The following six heuristic approaches were employed. Heuristics are criteria to select the next piece to be placed and the corresponding object to place it. The 1D and 2D cases of the BPP share the same selection heuristics. For the 2D BPP, additionally a placement heuristic has to be applied to find a solution.

1. First Fit Decreasing (FFD)
2. Filler + FFD
3. Best Fit Decreasing (BFD)
4. Djang and Finch with initial fullness of  $1/4$  ( $DJD_{1/4}$ )
5. Djang and Finch with initial fullness of  $1/3$  ( $DJD_{1/3}$ )
6. Djang and Finch with initial fullness of  $1/2$  ( $DJD_{1/2}$ )

Description of each heuristic is found in Section 2.6.1. All of these are single-pass constructive heuristics for the offline BPP. In preliminary experimentations (reported in Chapter 7), it has been found that  $DJD_{1/4}$ ,  $DJD_{1/3}$  and  $DJD_{1/2}$ , though similar, present a different behavior in different types of problem instances. For example, 2D instances with huge pieces are generally better solved by  $DJD_{1/4}$ . The three types of convex instances with the largest pieces (*Conv B*, *Conv G* and *Conv H*, see Table 4.3) are best solved by  $DJD_{1/4}$  (see Table 7.2). On the other hand, it is preferable to solve 2D regular instances (type *Conv I*) with  $DJD_{1/2}$ , rather than  $DJD_{1/4}$  and  $DJD_{1/3}$  (see Table 7.2). In this investigation, we set  $w = 1/20$  of the object or bin size as suggested in Chapter 7.

The heuristic Constructive Approach with Maximum Adjacency (CAD) was employed for finding the actual placement of the selected piece in a position inside the object for all the 2D instances. This is the only placement heuristic used and was chosen because of its good performance (see Section 7.4). This means that our heuristic repository  $H$  is now comprised of 6 single heuristics, which represents a huge reduction from the size of 40 handled in previous implementations (reported in Chapters 5 and 6). Chosen heuristics were selected from a larger set, because they produced the best single-heuristic results in a preliminary study. This ensures the quality of the single heuristic repository.

Previous studies had included in their heuristic repository every possible combination of several selection and placement heuristics considered, without performing any quality filter;

so, a very long list of heuristics comprises the heuristic repository [138, 143]. In those investigations, after the hyper-heuristics were built, most of the single heuristics were not employed when solving a large set of instances. The presence of bad-quality heuristics may delay the evolutionary process because it starts with an initial population with many bad-quality hyper-heuristics.

**Chromosomes.**- In initial population, number of blocks per chromosome is from 10 to 15 according to a uniform distribution. In this particular implementation, each block has 11 numbers: 10 for the representation of the instance state and the last one is a label, which identifies a single heuristic (selected from the 6 possible selection heuristics). The single heuristic is the corresponding action for a given state.

**Instances Testbed.**- In this investigation, the available 1417 testing instances were employing (see Section 4.2). The *Fu* instance from Table 4.1 was not employed here since it was always solved with 2 objects by every single heuristic.

**Fitness function.**- The objective function employed in this implementation is computed as explained in Section 3.4.

**Codification of Problem Instances.**- Each problem instance is codified as a text file. This process is explained in Section 3.7. 1D items are handled as 2D rectangles with a fixed width.

**Representation of the Instance State.**- The 10 features selected for representing every instance state were extracted employing the methodology explained in Chapter 6. This process is presented below. Previous implementations utilized only 8 features while now more features were significantly related with heuristic performance. It seems natural to increase this number now that we integrate 1D and 2D non-convex instances, though.

### 8.1.1 Developing a Problem-state Representation for the Testbed Instances

Each instance to be solved by the hyper-heuristic is characterized by a numerical vector that summarizes some of its relevant features. The most relevant features for our testbed instances are those related with single heuristic performance. A data-mining based methodology for establishing an adequate problem-state representation was proposed and explained in Chapter 6 [99, 101]. The general methodology comprises six steps that we applied as follows:

**Step 1.** Each instance is solved with each of the six single heuristics and its performance is computed with Equation 3.2.

**Step 2.** The single heuristics performance in each instance is considered as a vector in  $\mathbb{R}^6$ , where 6 is the size of the heuristics repository. We normalized this vector, dividing by its length (to get length of one). Unit vectors are used to indicate direction, that is, they aim in the direction of the better heuristic for each instance.

**Step 3.** All instances were grouped into homogeneous clusters according to their normalized performance measure for the 6 single heuristics. We chose the *k*-means clustering technique. In this procedure the number of clusters has to be provided by the user, however there exist some approaches for selecting a good number of clusters [38]. In our research, the number of clusters was chosen according to the Hartigan criteria, described by Chiang and Mirkin [38]; so, instances were grouped into eight homogeneous clusters depending on their

fitness in the six single heuristics. With the number of clusters chosen, 30 random initializations were run, and we decided to use the seed that produces clusters which minimize the total intra-cluster variance, or, the squared error function, given by:

$$V = \sum_{i=1}^k \sum_{x_j \in S_i} (x_j - \mu_i)^2 \quad (8.1)$$

where there are  $k$  clusters  $S_i$ ,  $i = 1, 2, \dots, k$ , and  $\mu_i$  is the centroid or mean point of all the points  $x_j \in S_i$ . Table 8.1 reports the number of instances associated to each cluster (cluster membership) for each instance type. Table 8.2 summarizes the cluster membership according to dimensionality and convexity. Instances in the same cluster have similar behavior when solved by the six heuristics considered. Note that many instance types are splitted into different clusters.

**Step 4.** This step consists of finding instance features that may be relevant to the ability of the heuristics to solve each instance [99]. Twenty three numerical features were computed for each instance. The first 19 are the same features listed in Section 6.2, where the methodology was applied for the first time. In the solution model implementation of this chapter, the following four features related with concavities are also included:

20. Percentage of non-convex pieces.
21. Average of the largest internal angle of all instance pieces.
22. Mean of the degree of concavity of the instance pieces (defined in Section 4.2.3).
23. Average of the proportion (area of piece)/(area of convex hull) for all instance pieces (defined in Section 4.2.3).

All items in 1D instances have only one dimension (height), so, their width has a variance of zero, meanwhile 2D instances will have a width variance greater than zero. For 1D instances, the area is computed assuming all items and bins have a fixed width, which means that area is proportional to height. All 1D items and 2D rectangles have rectangularity of 1.

**Step 5.** It may happen that some couple or small sets of features are highly correlated (positive or negative) since they carry almost the same information. Therefore, we deleted features 16, 17, 18 and 19 from the list of Step 4 and kept feature 10 (mean rectangularity) since this feature has an absolute Pearson correlation coefficient of at least 0.95 with the previous four features. The features deleted are related with horizontal/vertical edges, right angles and rectangularity which can be summarized computing the mean rectangularity. Besides, features 21 and 23 (related with concaveness) were deleted because they have an absolute Pearson correlation coefficient of at least 0.95 with feature 20 (percentage of non-convex pieces). Finally, from the 23-feature list generated in Step 4, we keep 17 features.

**Step 6.** Among the set of the 17 problem features, we selected a subset which better predicts the instances clusters. This selection was done through Multinomial Logistic Regression (MLR) [58]. The MLR model was run with all the features as independent variables and the cluster membership as the dependent variable. Taking all our testbed instances, we found 9 significant features. Instances in each cluster tend to have better performance in the same



Table 8.1: Number of instances associated to each cluster for all the instance types considered. The clusters are obtained according to the fitness of the six single heuristics selected.

Type	Cluster								Total
	C1	C2	C3	C4	C5	C6	C7	C8	
DB1 n1	41				4				45
DB1 n2	39				6				45
DB1 n3	34				11				45
DB1 n4	36				9				45
DB2 n1	16	6		1	6	1			30
DB2 n2	21	3		2	4				30
DB2 n3	13	3		3	11				30
DB2 n4	12	2		3	13				30
Waescher	12				5				17
Trip60			3	17					20
Trip120			1	19					20
Trip249				20					20
Trip501				20					20
Conv A	25				5				30
Conv B		3	11	10	6				30
Conv C	11	2	2	9	2	1	1	2	30
Conv D	25					3	1	1	30
Conv E	17	1			4	3		5	30
Conv F	25				5				30
Conv G	5	13	5	5	2				30
Conv H		5	14	7	4				30
Conv I	24				6				30
Conv J	28				1			1	30
Conv K	21	3		1	3	2			30
Conv L	18	2	3	2		3		2	30
Conv M	17	2	1		4	2	2	2	30
Conv N	26				4				30
Conv O	11	1		5	10	2	1		30
Conv P	16	4			10				30
Conv Q	12	5			3		8	2	30
Conv R	6	7		2	12	1	2		30
NConv A	25			1	2			2	30
NConv B	1	12	6	7	4				30
NConv C	9	4		2	12		2	1	30
NConv F	20				10				30
NConv H		13	6	8	3				30
NConv L	16		3	2	2		1	6	30
NConv M	9	1		5	13		2		30
NConv O	9	5		7	7		1	1	30
NConv S	15	1	2		7		2	3	30
NConv T	13	8	1		5		3		30
NConv U	7		5	3	9		4	2	30
NConv V	24	2	2		2				30
NConv W	16		2	2	7		2	1	30
NConv X	20	1	2	1	2		1	3	30
NConv Y	10	3		4	13				30
NConv Z	15	8		1	5		1		30
Total	720	120	69	169	253	18	34	34	1417

Table 8.2: Cluster membership for the 1D and 2D instances. According to fitness of the six single heuristics considered.

Cluster	1D	Convex 2D	Non Convex 2D	Total
C1	224	209	287	720
C2	14	58	48	120
C3	4	29	36	69
C4	85	43	41	169
C5	69	103	81	253
C6	1		17	18
C7		19	15	34
C8		19	15	34
Total	397	480	540	1417

single heuristics. We assume that this goes both ways, thus, those features that are more related with the clusters will also mostly influence instances to be better solved for one heuristic or another.

The nine significant features comprise the numerical representation. We added a tenth feature regarding the fraction of the instance total items that remain to be packed, so the GA learning process can have a sense of how advanced is the solution of a given instance (see Table 8.3). We performed a linear mapping to a fixed scale to all feature values used in the instance state representation making each possible feature value to fall inside the range from 0 to 1, so each numerical term has the same weight.

This numerical representation is capable of discriminating among the different categories of instances. When looking at features 4, 7 and 9 of the ten-value numerical representation of a given instance (see Table 8.3), it is possible to know what category of instance we have at hand. For example, only 1D instances have variance of width = 0; while a 2D regular instance has the following values: variance of width  $\neq 0$ , mean of rectangularity = 1 and mean of degree of concavity = 1.

## 8.2 Experimental Design

Two experiments were conducted sorting the 1417 available instances in two balanced training and testing sets and other two experiments swapped training and testing sets. Four experiments overall. The experiments are described below:

- **Experiment 1.** The training set is formed from the following instance types from Table 4.1: *DB1 n1* through *DB1 n4*, *Wäscher*, *Conv A* through *Conv I* and types *NConv A* through *NConv O*, totaling 707 instances. The rest of instance types comprises the testing set for this experiment.
- **Experiment 2.** This experiment is similar to Experiment 1, except that the training and testing sets are interchanged.

Table 8.3: Representation of the instance state.

Feature	Description
1	Number of pieces.
2	Mean area of remaining pieces.
3	Variance of the area of remaining instance pieces.
4	Mean of the rectangularity of remaining pieces.
5	Variance of the rectangularity of remaining pieces.
6	Mean of the height of the remaining pieces.
7	Variance of the width of the remaining pieces.
8	Fraction of remaining pieces in the instance whose area is above $1/2$ of the object area.
9	Mean of the degree of concavity of the remaining pieces.
10	Fraction of the instance total items remaining.

- **Experiment 3.** In this experiment every second instance from the testbed is selected. Therefore, half of instances of every available type are in the training set, totaling 709 instances. The rest of instances comprises the testing set for this experiment.
- **Experiment 4.** It is the same as Experiment 3, except that the training and testing sets are swapped.

Experiments were conducted with population size of 30, crossover probability of 1.0, mutation probability of 0.10, for 80 generations. These parameters worked well for the research presented in [99] and Chapter 6. Now, we increase the number of replicas for each experiment. For each of the four experiments, five GA processes were run. Overall, 20 hyper-heuristics were employed to measure the effectiveness of the model. For each complete run the two individuals with highest fitness were employed to solve the whole testing set and the best was selected as the hyper-heuristic of the run. The fitness of every individual of a GA run is an estimate since it is computed solving a sample of problem instances; that is why, to have a better sense of which individual may be best, we solve the whole testing set with the two best individuals from each GA run.

## 8.3 Results and Discussion

This section presents main results and some analysis in order to understand how the hyper-heuristic model works.

The hyper-heuristics generated have an average of 11.2 blocks. Since our heuristic repository has 6 single heuristics, some of them appear several times in a eleven-block hyper-heuristic. Besides, we note that every hyper-heuristic generated employed a combination of 2 to 5 single heuristics when solving a whole testing set. The six single heuristics considered were employed by at least one of the hyper-heuristics generated along the experiments.

Table 8.4: Hyper-heuristic generated in the first run of Experiment 1.

(Block)	Feature										Action
	1	2	3	4	5	6	7	8	9	10	
1	1.03	1.33	0.8	0.49	0.64	0.22	0.81	0.43	0.91	0.65	0 (FFD)
2	-0.08	0.16	0.77	0.17	0.43	0.74	-0.34	0.37	0.92	0.3	1 (Filler)
3	1.06	0.89	0.4	-0.36	0.7	0.51	-0.03	0.84	1.39	0.36	2 (BFD)
4	0.34	0.91	0.56	0.27	0.41	0.93	0.87	0.82	0.91	0	2 (BFD)
5	1.08	1.25	0.83	1.26	0.51	0.04	0.49	0.02	0.7	0.91	0 (FFD)
6	0.57	0.1	0.46	0.52	0.67	0.39	0.87	0.44	-0.61	0.43	3 (DJD <sub>1/4</sub> )
7	0.52	0.8	1.14	0.34	0.52	0.33	0.8	1.05	0.17	-0.42	4 (DJD <sub>1/3</sub> )
8	0.7	1.08	0.87	-0.23	0.52	-0.59	0.89	0.23	0.55	0.31	2 (BFD)

As an example, Table 8.4 shows the hyper-heuristic generated by the first run of Experiment 1. It has 8 blocks and may employ up to 5 different single heuristics (actions) when solving a given problem instance. Features number 1 to 10 are described in Table 8.3. In this example, when the hyper-heuristic is used to solve the Experiment 1 testing set, it only employs two different actions: heuristics 1 and 3 (Filler and DJD<sub>1/4</sub>). This is because the other blocks represent problem states that were not reached by the instances solved. Most of the 710 problem instances were solved employing a combination of heuristics 1 and 3, and only 39 solutions of 2D instances were constructed employing one single heuristic. Even though, an actual representation of a problem state has values only inside the range from 0 to 1, we permit values to fall outside the range from 0 to 1 in a hyper-heuristic (see Table 8.4), which means that some points labelled with single heuristics could lie right outside the unit hypercube shown in Figure 3.3.

### 8.3.1 Comparison Against the Best Single Heuristic

Each hyper-heuristic generated was used to solve the testing set of instances of the experiment where it comes from. Results are shown in Table 8.5. Figures in cells indicate the percentage of problems that employs a particular number of extra objects (left column) when compared against results provided by the best single heuristic for each instance. We present results averaging the 5 runs of each experiment as well as the results for the best hyper-heuristic per experiment. For example, when solving the testing set instances of Experiment 1, the average performance of five hyper-heuristics beats the best single heuristic in 0.6% of the testing cases, while the best hyper-heuristic per instance required one object less in 2.7% of instances (comparing with the best result obtained for any of the six single heuristics). For Experiment 2, the best hyper-heuristic produces the same results that the best single heuristic 100% of times; which means that solving each testing instance with the 5 hyper-heuristics generated and then choosing the best result, or employing the 6 single heuristics and selecting the best, will need the same number of objects. For Experiment 1, however, it is slightly better to use the best of the 5 hyper-heuristics generated.

Table 8.5: Number of extra objects obtained by hyper-heuristics and single heuristics compared against results of the best single heuristic for each instance (percentage of cases). Zero values are displayed as blank cells.

<b>Experiment 1</b>								
Extra objects	<b>Hyper-heuristics</b>		<b>Single heuristics</b>					
	Average	Best	FFD	Filler	BFD	DJD <sub>1/4</sub>	DJD <sub>1/3</sub>	DJD <sub>1/2</sub>
$\leq -2$		0.7						
-1	0.6	2.7						
0	86.2	94.9	62.1	63.0	64.5	96.1	93.0	72.0
1	13.2	1.7	23.0	22.0	20.7	3.9	6.6	14.8
2			3.9	4.1	3.8		0.4	2.8
3			1.1	1.1	1.1			0.8
$\geq 4$			9.9	9.9	9.9			9.6
<b>Experiment 2</b>								
Extra objects	<b>Hyper-heuristics</b>		<b>Single heuristics</b>					
	Average	Best	FFD	Filler	BFD	DJD <sub>1/4</sub>	DJD <sub>1/3</sub>	DJD <sub>1/2</sub>
$\leq -2$								
-1								
0	92.6	100	70.2	69.4	70.6	96.3	95.8	72.8
1	7.4		21.1	21.8	20.5	3.5	4.2	19.5
2			6.2	6.2	6.4	0.1		6.1
3			2.4	2.4	2.4			1.6
$\geq 4$			0.1	0.1	0.1			
<b>Experiment 3</b>								
Extra objects	<b>Hyper-heuristics</b>		<b>Single heuristics</b>					
	Average	Best	FFD	Filler	BFD	DJD <sub>1/4</sub>	DJD <sub>1/3</sub>	DJD <sub>1/2</sub>
$\leq -2$		0.3						
-1		0.8						
0	94.6	96.9	66.1	66.1	67.2	96.0	94.1	72.2
1	5.4	2.0	21.6	21.6	20.3	3.8	5.6	17.1
2			5.4	5.4	5.5	0.1	0.3	4.8
3			2.0	2.0	2.0			1.3
$\geq 4$			4.9	4.9	4.9			4.7
<b>Experiment 4</b>								
Extra objects	<b>Hyper-heuristics</b>		<b>Single heuristics</b>					
	Average	Best	FFD	Filler	BFD	DJD <sub>1/4</sub>	DJD <sub>1/3</sub>	DJD <sub>1/2</sub>
$\leq -2$		0.1						
-1		0.8						
0	91.7	97.9	66.1	66.3	67.8	96.3	94.6	72.6
1	5.1	1.1	22.4	22.1	20.9	3.7	5.2	17.2
2	1.6		4.8	4.9	4.7		0.1	4.1
3	0.6		1.6	1.6	1.6			1.1
$\geq 4$	1.1		5.1	5.1	5.1			4.9

In Table 8.6, we report the average number of extra objects delivered by the best hyper-heuristic per experiment, compared against the number of objects employed by the best single heuristic for each instance. For experiments 1, 3 and 4, the best hyper-heuristic for 1D instances deliver fewer objects than the correspondent best single heuristic (numbers are negative for 1D instances). On average, the best hyper-heuristic of the 5 runs solved 1D instances employing 0.028 objects less than the best of the 6 single heuristics. While, for the 2D case, the best hyper-heuristic employs an average of 0.003 and 0.005 objects more for instances with convex and non-convex pieces respectively.

Table 8.6: Average number of extra objects delivered by the best hyper-heuristic, compared against results of the best single heuristic for each instance.

Experiment	1D	Convex 2D	Non Convex 2D
1	-0.090	0.007	-0.008
2	0	0	0
3	-0.005	0.011	0.008
4	-0.015	-0.007	0.021
Average	-0.028	0.003	0.005

For most cases, the best hyper-heuristic requires the same number of objects than the best single heuristic (higher percentages in Table 8.5 are in the 0-object row). Therefore, we employed the non-parametric Mann-Whitney U statistical test for means comparison of extra objects between 1D and 2D cases. We want to know if the hyper-heuristic model performance is different for 1D and 2D instances. For Experiment 1, the extra number of objects delivered by the best hyper-heuristic is statistically different for 1D and for 2D instances ( $p$ -value = 0.001). For the rest of the experiments, the difference is not significant between 1D and 2D BPP. When we perform a comparison of means test between results for convex and non-convex 2D instances, we found that there is a significant difference only in Experiment 4 ( $p$ -value = 0.016). In conclusion, most of the experiments show that hyper-heuristics performance is not statistically different for the distinct categories of BPP considered.

### 8.3.2 Analyzing Results per Instance Category

There is a correspondence between the category of problem instances and the single heuristics more often employed. This is what we expected because different categories of instances have different numerical representations; so, the hyper-heuristics suggests different single heuristics to apply. In Table 8.7 we illustrate this fact averaging all the runs of Experiment 1. For 1D instances, the Filler heuristic was employed 29.1% of the times, while this heuristic was chosen only 7.5% of the times when solving 2D convex instances. We ran a test of contingency table with the  $\chi^2$  statistic to verify this, concluding that usage of single heuristics is indeed related to instance category ( $p$ -value < 0.001). Also, there is a significant difference in the employment of single heuristics between the two types of 2D instances considered (convex and non-convex). We arrive at the same conclusion when considering the other three experiments.

Table 8.7: Percentage of selection of each single heuristic when solving the testing set with hyper-heuristics of Experiment 1.

	1D	Convex 2D	Non Convex 2D
FFD	21.6	3.1	2.1
Filler	29.1	7.5	15.3
BFD		8.9	9.5
DJD <sub>1/4</sub>	43.2	63.7	60.7
DJD <sub>1/3</sub>	4.5	15.6	12.1
DJD <sub>1/2</sub>	1.7	1.3	0.3

This analysis shows that we have found hyper-heuristics that are able to solve properly different kind of problem instances without a human hand to choose different single heuristics for different cases. It is worth mentioning that the heuristic DJD<sub>1/4</sub> was the most used in solving all the instance types, which suggests that this is a very effective and robust heuristic.

### 8.3.3 Comparing Results for Convex and Non-convex Instances

There are 240 convex instances in the testbed that have also their non-convex version. They are instances types *Conv A*, *Conv B*, *Conv C*, *Conv F*, *Conv H*, *Conv L*, *Conv M* and *Conv O* from Table 4.1. Their respective non-convex version have exactly the same pieces, except for those that were split to generate concaveness (see algorithm from section 4.2.4 and Figure 4.5). Therefore, optimum solutions of the non-convex instances have the same number of objects than their respective convex instances, all with objects filled up to 100%. For the 240 convex instances, the number of pieces goes from 28 to 40. The number of pieces chosen to be split in each of these convex instances goes from 5 to 24. We want to compare how single heuristics and hyper-heuristics solve problem instances that have several pieces in common. These results are summarized in Table 8.8. Values in cells indicate percentage of instances where the best single heuristic (or the best hyper-heuristic) have employed fewer, equal or more objects to solve the non-convex version compared with the number of objects employed to solve each convex instance. For example, the best single heuristic per instance employed the same number of objects in 65% of the 240 non-convex instances than the number of objects employed for their respective convex version. In 0.4% of the instances (which means only one case), the best single heuristic solved a non-convex instance employing fewer objects than its convex version. In the rest of the cases, approximately one-third, solving the non-convex instance requires more objects than solving the convex instance. Either instances are solved by the best single heuristic or by the best hyper-heuristic of any experiment, results are basically the same: in about one-third of the cases, solving a problem instance with split pieces requires more objects than solving the original instance. This may be due to the fact that our general-purpose methodology is not intended to match a piece with the concavity of other piece where it fits. We are dealing with a combination of fast single-pass constructive heuristics which means that all pieces have only one opportunity to couple with the pieces that perfectly complements them.

Table 8.8: Solving non-convex instances compared when solving their convex version. Percentage of cases when non-convex instances require fewer, equal and more objects than convex instances. Instances are solved by the best single heuristic and by the best hyper-heuristic from experiments 1 through 4.

	Best single heuristic	Best hyper-heuristic			
		Exp 1	Exp 2	Exp 3	Exp 4
Fewer objects	0.4	1.3	0.8	1.3	0.4
Same number of objects	65	64	63	64	63
More objects	34	35	37	35	37

### 8.3.4 Alternation of Single Heuristics

When a hyper-heuristic solves a given problem instance, it computes the problem state after every heuristic application. Each heuristic application places exactly 1 piece, except  $DJD_{1/4}$ ,  $DJD_{1/3}$  and  $DJD_{1/2}$  that may place 1, 2 or 3 pieces. Most of the times, successive computations of problem states are similar, since the application of one or few heuristics does not change much the problem state leading to the same block in the chromosome (see Figures 3.2 and 3.3). Therefore, it is common to choose the same heuristic several times sequentially. Moreover, several blocks may have the same heuristic, as it happens in the hyper-heuristic shown in Table 8.4. Therefore, it is possible to select the same single heuristic even when changing blocks in the hyper-heuristic solution process. Averaging all our experiments, 46.9% of instances with up to 50 pieces were solved using one single heuristic; and 27.6% of these instances have only one change of single heuristic when building the solution. This means that one heuristic was employed for placing the first pieces and then, another heuristic was chosen to finish placing the rest of the pieces. 10.0% of instances with 50 pieces or less involved 2 heuristic changes when solved by a hyper-heuristic. By contrast, there are few instances that were solved with up to 20 heuristic changes. Table 8.9 shows the results of the analysis of heuristic alternation. Note that several heuristic changes may imply that the hyper-heuristic is returning to single heuristics previously employed in the same problem instance.

Table 8.9: Percentage of single heuristic changes when solving all testing sets.

Heuristic changes	Number of pieces				All instances
	up to 50	51 - 100	101 - 200	201 - 500	
0	46.9	48.7	34.6	33.4	45.4
1	27.6	28.0	38.7	37.2	29.3
2	10.0	7.7	8.9	9.7	9.2
3	8.4	7.4	7.6	6.6	7.9
4	2.8	3.2	3.1	4.6	3.1
$\geq 5$	4.4	4.9	7.1	8.5	5.1



We are interested in exploring whether the quality of solutions is related with the number of heuristic changes performed during the solution process. Table 8.10 summarize results for all experiments to show this. Hyper-heuristics perform an average of 2.7 heuristic changes while solving instances that require one object less than the best single heuristic. The same hyper-heuristics make 1.1 heuristic changes when solving instances whose solution is the same that the best single heuristic. For those cases where hyper-heuristics solutions require more objects, more heuristic changes are done. This is, hyper-heuristics perform more heuristic changes with the best solutions found as well as with the worse solutions found. This makes sense when we consider that a hyper-heuristic that makes few heuristic changes will produce a solution similar to one of the single heuristics. Hyper-heuristics find different solutions when they *dare* to combine a greater number of single heuristics. In general, with more changes between single heuristics, a better solution may emerge (with the risk of producing a worse solution, though).

Table 8.10: Average of heuristic changes according to the number of extra objects against results of best single heuristic.

Extra objects against best single heuristic	Number of pieces				<b>All instances</b>
	up to 50	51 - 100	101 - 200	201 - 500	
$\leq -2$				2.3	2.3
-1	2.4	3.3	4.0	2.6	2.7
0	1.1	1.0	1.2	1.2	1.1
1	1.6	1.7	2	2.4	1.8
2	5.3	1.5	3.6	4.1	3.2
3	1.5	5.8	4.9	3.5	4.5
$\geq 4$		4.0	5.3	8.2	8.2

Table 8.11 shows how long are the sequences of the same single heuristic before changing to another heuristic. For example, in instances with up to 50 pieces, these sequences have an average length of 16.9. This means that the same heuristic is applied an average of 16.9 times after the hyper-heuristic changes to another single heuristic. For further research, we plan to test the use of the same single heuristic a user-given number of times before recomputing the problem state. This may reduce even more the time of computation keeping good results.

Also, we analyzed which sequences of single heuristics were performed during our experiments. That is, which heuristics tend to follow which others. For example, for 1D instances, 23.1% of all heuristic changes were about heuristic  $DJD_{1/3}$  followed by  $DJD_{1/4}$  (see Table 8.12).  $DJD_{1/4}$  is the first heuristic in 40.8% of all heuristic changes. Tables 8.13 and 8.14 show the corresponding results for 2D convex and non-convex instances respectively.

Notice that Tables 8.12, 8.13 and 8.14 are highly asymmetric matrices. For instances from all types, heuristic BFD almost exclusively goes before and after heuristics  $DJD_{1/4}$  and  $DJD_{1/3}$ . This means that heuristic BFD almost never makes pair with heuristics FFD, Filler and  $DJD_{1/2}$  (see Tables 8.12, 8.13 and 8.14). Moreover, heuristics FFD, Filler and BFD never follow each other when solving 2D instances. These three heuristics place a piece one at a

Table 8.11: Average length of single heuristic sequences when applying our hyper-heuristic solution model.

Number of pieces	Average length of heuristics runs
up to 50	16.9
51 - 100	31.8
101 - 200	83.6
201 - 500	205.3
All instances	41.8

Table 8.12: Percentage of sequences of single heuristic pairs when solving 1D instances in all testing sets.

From heuristic	To heuristic						Total
	FFD	Filler	BFD	DJD <sub>1/4</sub>	DJD <sub>1/3</sub>	DJD <sub>1/2</sub>	
FFD		0.1		0.8	2.0		2.9
Filler					8.6	0.7	9.3
BFD	0.6			1.8	0.6		3.0
DJD <sub>1/4</sub>	1.5	6.7	5.7		10.7	16.2	40.8
DJD <sub>1/3</sub>	4.4	6.9	1.1	23.1		0.4	35.9
DJD <sub>1/2</sub>		0.3		7.5	0.4		8.2
Total	6.5	14.0	6.8	33.2	22.3	17.3	100

time, while the remaining heuristics (DJD<sub>1/4</sub>, DJD<sub>1/3</sub> and DJD<sub>1/2</sub>) place groups of 1, 2 or 3 pieces. That is, 1-piece heuristics always alternate with the DJD heuristics.

### 8.3.5 Time Complexity

We ran the experiments of this chapter on a 1.66 GHz PC with 1.98 GB of RAM. Solving an instance with a hyper-heuristic is faster than solving it with each of the single heuristics and then choosing the best result. Once the hyper-heuristic is generated, it solves each instance in 21 seconds on average (see Table 8.15). Although the process for generate the hyper-heuristic is much slower, since it requires to solve many instances. With the initial population, 5 instances are given to each of the 30 individuals. Later, only the 2 new offspring receive 5 instances, all other 28 chromosomes solve only one new instance to update its fitness. We perform  $30 \times 5 = 150$  evaluations in the first generation and  $2 \times 5 + 28 \times 1 = 38$  evaluations in each of the following 79 generations. That is, a GA run involves  $150 + 79 \times 38 = 3152$  evaluations. If each evaluation is done in 21 seconds on average, the total time for all evaluations is 18.4 hours which is about the time that we observed experimentally for each GA run. Besides the application of the single heuristics in training instances, the GA employs some computational resources in computing the numerical representation state every time a heuristic places pieces

Table 8.13: Percentage of sequences of single heuristic pairs when solving 2D convex instances in all testing sets.

From heuristic	To heuristic						Total
	FFD	Filler	BFD	DJD <sub>1/4</sub>	DJD <sub>1/3</sub>	DJD <sub>1/2</sub>	
FFD				2.6	0.2	3.8	6.6
Filler				4.7	4.6		9.3
BFD				0.8	2.7		3.5
DJD <sub>1/4</sub>	7.1	12.6	2.9		8.0	19.0	49.6
DJD <sub>1/3</sub>	0.6	4.3	6.7	10.5		0.5	22.6
DJD <sub>1/2</sub>	2.2			5.6	0.5		8.3
Total	9.9	16.9	9.6	24.2	16.0	23.3	100

Table 8.14: Percentage of sequences of single heuristic pairs when solving 2D non-convex instances in all testing sets.

From heuristic	To heuristic						Total
	FFD	Filler	BFD	DJD <sub>1/4</sub>	DJD <sub>1/3</sub>	DJD <sub>1/2</sub>	
FFD				2.6	0.2	3.1	5.9
Filler				3.7	6.3	0.5	10.5
BFD				0.9	1.9		2.8
DJD <sub>1/4</sub>	6.1	9.9	3.5		9.4	19.3	48.2
DJD <sub>1/3</sub>	0.8	5.9	5.2	10.5		0.5	22.9
DJD <sub>1/2</sub>	1.8	0.1	0.1	7.4	0.5		9.9
Total	8.7	15.9	8.8	25.1	18.3	23.4	100

when solving a problem instance.

The six single heuristics considered solve instances with a huge variety of time length. For example, the fastest single heuristic, FFD, solves 1D instances in 0.2 seconds per case, in average; while DJD<sub>1/4</sub> is the most time consuming heuristic averaging 24.8 seconds per 1D instance (see Table 8.15). The best single heuristic may be different for each case. Moreover, for many cases the smallest number of objects is got by several of the six heuristics. We averaged the recorded time employed by all single heuristics that got the smallest number of objects per instance. Table 8.15 shows that best single heuristics employ more time than the average heuristic. For example, for 1D instances, the best of the single heuristics employed 21.1 seconds per case.

The last two columns of Table 8.15 regard hyper-heuristics. Each instance from our testbed belonged to a testing set in two out of the four experiments performed. For example, the first instance of the set was in the training set of experiments 1 and 3; so, it belongs to experiments 2 and 4 testing sets (see Section 8.2). Hyper-heuristics were tested solving instances in the testing sets only. Five runs were performed for each experiment, meaning that 10 hyper-heuristics solved each instance. The next to the last column from Table 8.15

Table 8.15: Average computational time (in seconds) per category of instances.

	FFD	Filler	BFD	DJD <sub>1/4</sub>	DJD <sub>1/3</sub>	DJD <sub>1/2</sub>	Simple heuristics		Hyper-heuristics	
							Average	Best	Average	Best
1D	0.2	29.3	5.1	24.8	24.0	24.1	17.9	21.1	21.9	21.8
Convex 2D	2.5	14.2	2.5	18.5	18.5	18.4	12.4	12.8	20.7	20.7
Non Convex 2D	2.9	9.2	3.5	18.3	18.3	18.3	11.8	12.9	20.6	20.6
Total	2.0	16.7	3.6	20.2	20.0	20.0	13.7	15.1	21.0	21.0

average the 10 hyper-heuristic used to solve each problem instance, while the last column average the time employed only for those hyper-heuristics that got the smallest number of objects found by hyper-heuristics (for each case, several of the 10 hyper-heuristics may have obtained the lowest number of objects). It is worth noting that time obtained by the average hyper-heuristic is very similar to the time obtained by the best hyper-heuristic per case, and both times are larger than those obtained by any of the single heuristics. Hyper-heuristics compute the numerical state after each application of a heuristic. This may explain why hyper-heuristics employ more time than any of the single heuristics. Nevertheless, Table 8.5 shows that results from an average hyper-heuristic are better than average results from single heuristics. Table 8.16 shows the averages of computational time per instance type.

Number of pieces is indeed related with the computational time employed when solving a problem instance with any of the 6 heuristics. Figures 8.1 and 8.2 plot the number of pieces against the computational time for 1D instances and 2D instances respectively. In the three graphs regarding the DJD heuristics for the 1D instances, there are two hard instances from type *Wäscher* with sizes 228 and 239 pieces that employed a large amount of time for being solved compared with other 1D instances of similar size.

Even though we observe that the larger the instance, the more time required to solved, this relation is not perfect. There are some other variables that may matter. For example, the pieces size, the variability of pieces size, the number of pieces sides and the irregularity of the pieces may have an impact in the execution time. For the 1D instances from types *Trip60*, *Trip120*, *Trip249* and *Trip501* all these factors remain almost constant; because the optimum solution of these instances employ exactly 3 items per bin, which means that pieces sizes are similar across all the 80 instances from these types. Besides, the 1D instances do not present variability in shapes nor irregularity issues. Figure 8.3 plots the number of pieces against the computational time of the DJD<sub>1/3</sub> heuristic for each instance. A cubic polynomial with intercept in the origin (Time =  $0.0018x^3 - 0.5533x^2 + 37.506x$ , where  $x$  = number of pieces) fits very good the points in the plot ( $R^2 = 0.9991$ ). We tried a quadratic polynomial obtaining good results, but not as good as the results from cubic polynomial, though.

Figure 8.4 plots the number of pieces against the computational time employed by the average of the hyper-heuristics used in each of the instances. The outlier in the 2D graph corresponds to a 45-piece instance from the type *NConv F* with many small pieces, since its optimum is only 2 objects with zero waste. There are other instances with these characteristics, but the hyper-heuristics did not employ in them such a long time.

Table 8.16: Average computational time (in seconds) per type of instances.

	FFD	Filler	BFD	DJD <sub>1/4</sub>	DJD <sub>1/3</sub>	DJD <sub>1/2</sub>	Simple heuristics		Hyper-heuristics	
							Average	Best	Average	Best
DB1 n1	0.0	0.0	0.0	0.1	0.2	0.1	0.1	0.1	0.7	0.7
DB1 n2	0.1	0.1	0.1	0.5	0.5	0.5	0.3	0.3	1.1	1.1
DB1 n3	0.2	1.2	0.5	2.6	2.7	2.6	1.6	1.9	3.4	3.4
DB1 n4	0.5	106.7	39.3	115.3	108.7	109.1	79.9	90.8	99	98.1
DB2 n1	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	0.7	0.7
DB2 n2	0.1	0.2	0.1	0.3	0.3	0.3	0.2	0.2	0.9	0.9
DB2 n3	0.2	1.3	0.2	1.2	1.2	1.2	0.9	1	2.3	2.3
DB2 n4	0.6	105.7	2.3	61.6	61	60.7	48.6	56.2	54.3	54.4
Wäscher	0.5	0.8	0.4	21.8	21.8	21.8	11.2	11.8	26.7	26.6
Trip60	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5
Trip120	0.0	0.3	0	0.2	0.2	0.2	0.2	0.2	0.8	0.8
Trip249	0.1	4.5	0.2	3.1	3.1	3.1	2.4	3.1	4.3	4.1
Trip501	0.2	172.4	7.2	109.5	109.5	110.6	84.9	109.5	84.4	85.1
Conv A	0.9	2.9	0.9	9.0	9.0	8.9	5.3	5.3	7.7	7.7
Conv B	0.1	0.1	0.1	0.2	0.2	0.2	0.2	0.2	0.7	0.7
Conv C	1.0	1.7	1.3	3.8	3.9	3.9	2.6	3	6.2	6.3
Conv D	8.3	43.1	7.1	59.9	59.8	59.8	39.7	39.5	61.8	61.9
Conv E	8.9	81.7	8.3	54.8	54.9	54.9	43.9	43.3	58.5	57.9
Conv F	1.2	7.2	1.3	12.3	12.4	12.4	7.8	7.8	12.8	12.8
Conv G	0.2	0.2	0.2	0.9	0.9	0.9	0.5	0.8	1.4	1.4
Conv H	0.2	0.1	0.2	0.3	0.3	0.3	0.2	0.3	0.8	0.8
Conv I	0.8	2.5	0.7	6.4	6.4	6.4	3.9	3.9	13.0	13.0
Conv J	2.6	11.3	2.6	25.3	25.3	25.2	15.4	15.4	28.2	28.2
Conv K	1.6	4.7	1.5	8.0	8.1	8.0	5.3	5.5	10.8	10.7
Conv L	1.2	5.2	1.2	7.4	7.4	7.4	5.0	5.8	10.2	10.4
Conv M	2.2	5.1	2.5	9.7	9.6	9.6	6.4	7.3	15.2	15.2
Conv N	5.8	69.2	5.9	86.1	86	85.9	56.5	56.5	93.4	93.4
Conv O	0.5	0.2	1.0	1.2	1.2	1.2	0.9	1.1	1.5	1.5
Conv P	4.5	12.9	6.1	21.3	21.3	21.4	14.6	15.8	27.2	26.3
Conv Q	1.5	1.2	1.2	7.6	7.6	7.6	4.4	5.9	0.9	0.8
Conv R	3.5	5.4	3.8	17.9	18.2	17.9	11.1	12.7	23.3	23.1
Nconv A	3.9	16.8	3.3	33.4	33.4	33.4	20.7	20.9	28.1	28.3
Nconv B	2.5	3.4	3.1	16.7	16.8	16.7	9.9	14.1	21.4	21.4
Nconv C	5.5	14.1	5.3	22.4	22.4	22.4	15.3	15	30.1	29.9
Nconv F	3.7	28.3	3.7	49.6	49.8	49.8	30.8	30.8	48.8	48.8
Nconv H	2.0	2.7	2.7	17.5	17.5	17.7	10.0	17.4	20.3	19.9
Nconv L	4.3	24.3	4	39.3	39.2	39.1	25	24.2	36.6	37.1
Nconv M	5.8	22	5.9	34.1	34	34.1	22.6	26.8	43.0	43.1
Nconv O	2.2	2.5	3.9	9.9	9.9	9.9	6.4	7.5	12.0	12.1
Nconv S	0.7	2.1	0.9	3.0	3.0	3.0	2.1	2.2	4.4	4.3
Nconv T	0.9	0.5	2.4	1.8	1.8	1.8	1.5	1.6	2.1	2.1
Nconv U	1.4	2.2	2	7.1	7.1	7.1	4.5	4.9	9.2	9.2
Nconv V	0.2	0.1	0.3	0.2	0.2	0.2	0.2	0.2	0.8	0.8
Nconv W	0.6	1.0	0.5	1.8	1.8	1.8	1.3	1.4	3.7	3.8
Nconv X	2.6	8.3	2.1	14.8	14.8	14.8	9.6	9.5	16.8	16.8
Nconv Y	4.0	10.7	4.3	18.7	18.7	18.7	12.5	12.8	25.0	24.7
Nconv Z	6.5	8.2	12.4	22.8	22.8	22.8	15.9	16.8	27.0	27.0

## 8.4 Summary

In this research we applied an evolutionary approach to find a rule (hyper-heuristic) that combines single heuristics in such a way that it is able to solve efficiently a wide range of 1D and 2D Bin Packing Problem instances with good results and without parameter tuning. Among the 2D instances, there are rectangles, convex and non-convex polygons. Although all these are packing problems they are of very different nature. As a main conclusion, the research reported in this chapter has shown that the proposed evolutionary approach is able to generalize a solution procedure to a range of problem types.

For some of the instances, hyper-heuristics achieve better results than the best of the single heuristics showing that combination of single heuristics may outperform any of the single heuristics considered separately. These cases justify the existence of hyper-heuristics beyond any simple heuristic, since for some applications, any reduction in material is extremely valuable. Although, the most frequent case is the hyper-heuristic that produces the same result than the best single heuristic. This is also beneficial, as the choice of best heuristic varies from instance to instance, so the hyper-heuristics are definitely preferable to selecting any one heuristic for all problem instances.

We found that hyper-heuristics tend to choose different single heuristics for different kinds of instances. This is a sign that the evolutionary process has found that distinct instances states are more suitable to be solved with different single heuristics.

Next chapter uses a knowledge discovery approach, to seek insights into the relationships between problem structure and the effectiveness of heuristics and hyper-heuristics.

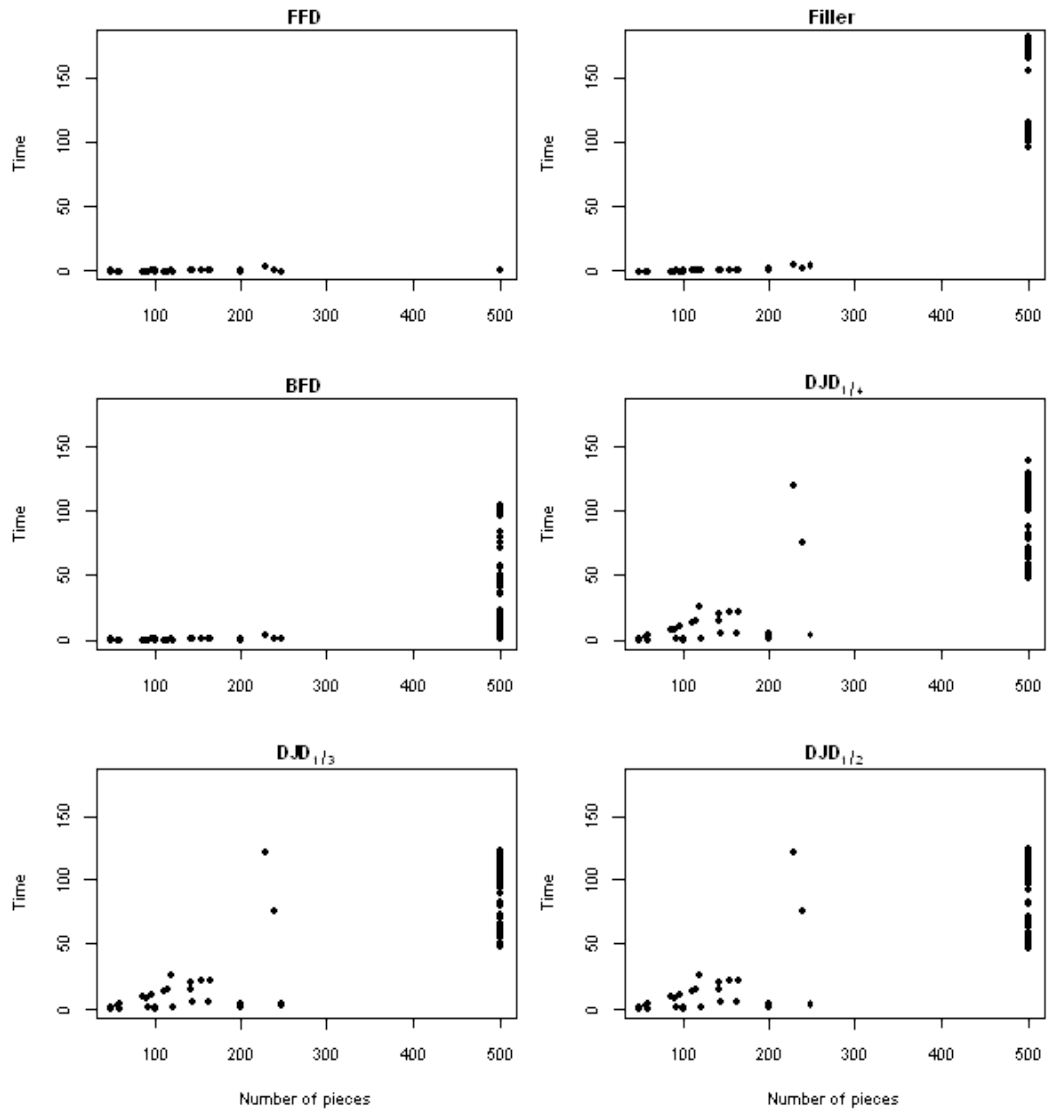


Figure 8.1: Number of pieces vs. computational time for the 397 one-dimensional instances. Six single heuristics.

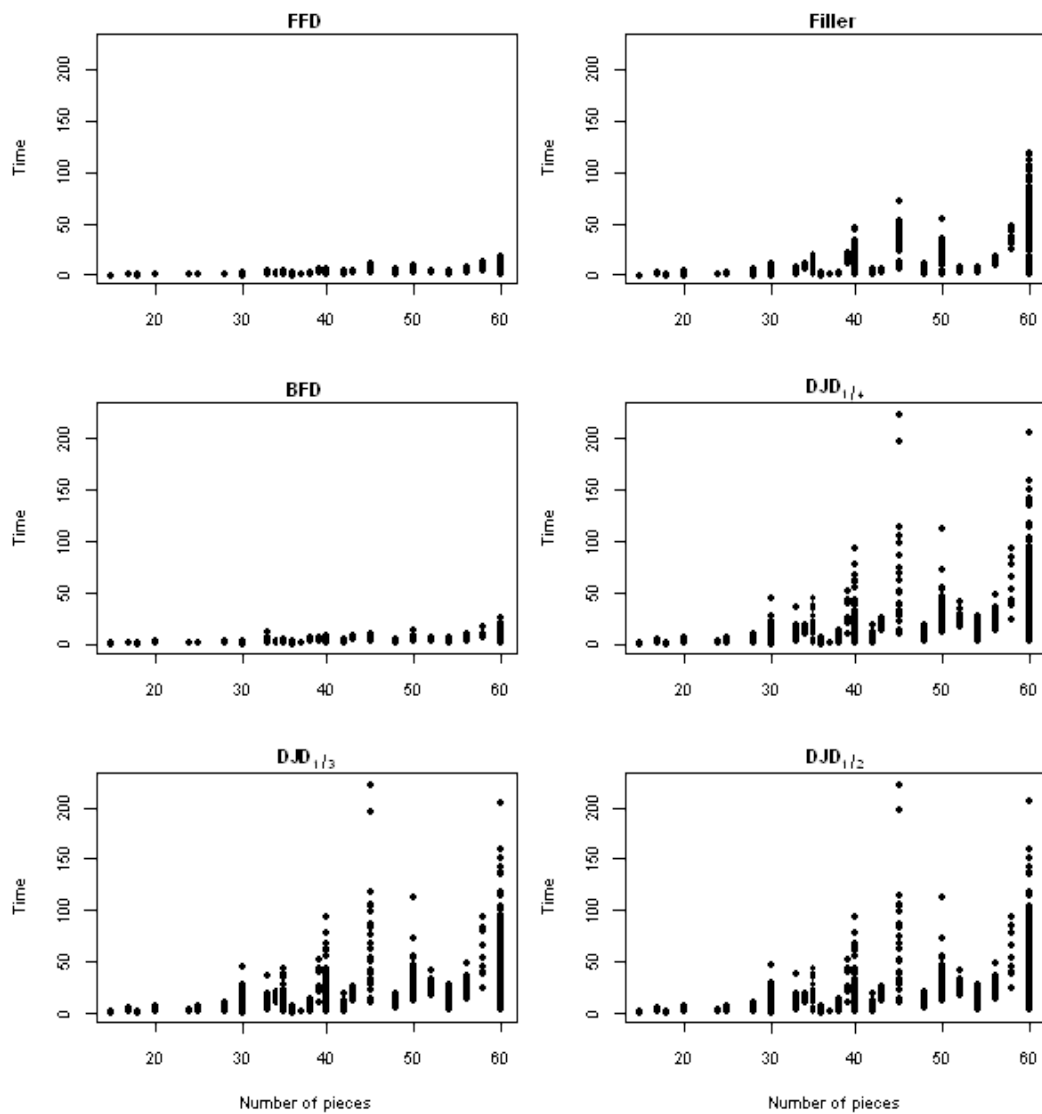


Figure 8.2: Number of pieces vs. computational time for the 1020 two-dimensional instances. Six single heuristics.



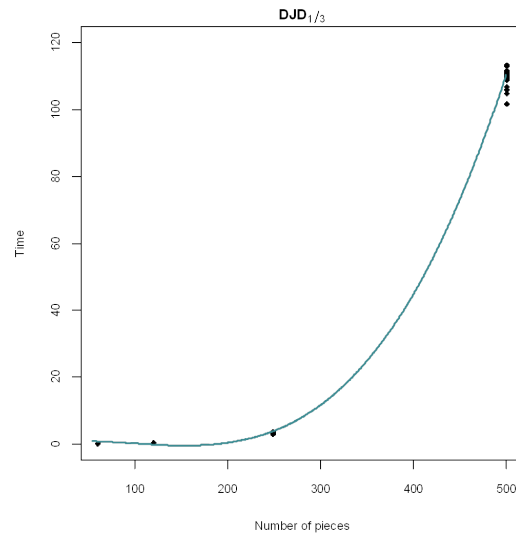


Figure 8.3: Number of pieces vs. computational time for the 80 one-dimensional instances of types *Trip60*, *Trip120*, *Trip249* and *Trip501*. Heuristic  $DJD_{1/3}$ . The line is the graph of the cubic polynomial which fits best all the points.

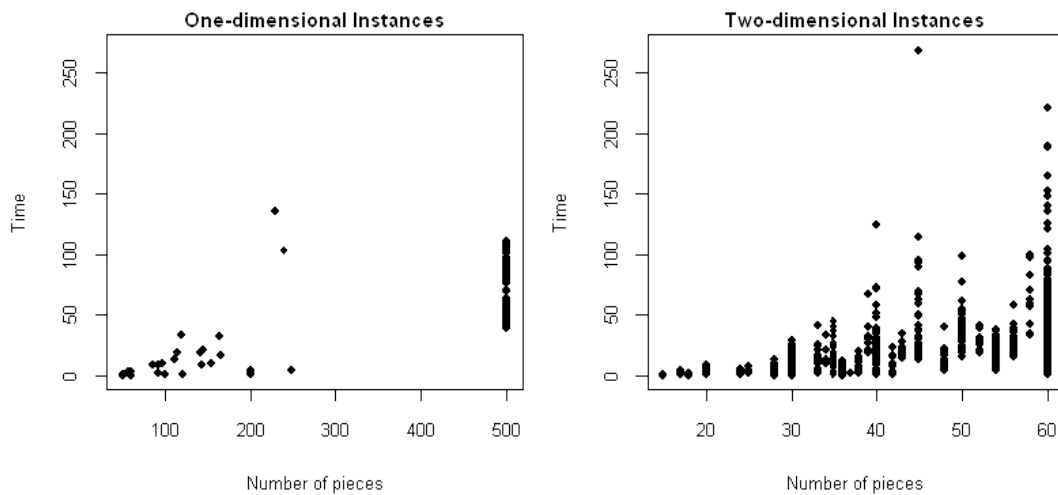


Figure 8.4: Number of pieces vs. computational time for the average of the hyper-heuristics tested in each instance. 1D and 2D instances.



# Chapter 9

## A Deeper Understanding of the BPP Structure

The goal of the investigation presented in this chapter is to gain a deeper understanding of the structure of the bin packing problem and how this structure impacts upon algorithm performance. A knowledge discovery approach is used to reveal the problem features or combination of features that influence the performance of bin packing heuristics and hyper-heuristics. This research is inspired in some ideas related to the meta-learning field (see Section 2.11).

Using the Principal Component Analysis method, the problem instances, characterized by 9 features, are visualized in two dimensions. These 9 features were selected from a larger set of 23 by a method that selects the subset of features that are more strongly related with algorithm performance. Different combinations of features characterize instances in each section of the 2D graph produced by the Principal Component Analysis method, and heuristic performance is overlaid over the 2D graph. This visualization approach reveals new knowledge on the relationship between bin packing problem features and heuristic and hyper-heuristic performance.

In previous experiments, we made 2D scatter plots for all possible pairs of variables and did not find clear patterns for heuristic performance. The PCA plots involve 9 variables at the same time and some patterns may be seen. This chapter proposes using Principal Component Analysis (PCA) for visualizing  $n$ -dimensional data related to bin packing problems. The goal is to improve our understanding of the problem structure and its relationship with algorithm performance. PCA is a mathematical algorithm that reduces the dimensionality of the data while retaining most of the variation in the dataset. It accomplishes this reduction by converting a set of observations into a set of values of uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has as high a variance as possible (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it be orthogonal to (uncorrelated with) the preceding components. Observations can then be plotted, making it possible to visually assess similarities and differences between observations and determine whether observations can be grouped [124]. In this study, PCA graphs are used to visually assess the problem feature combinations that are mostly related with an improved algorithm performance. The ultimate goal is to inform the design of effective heuristics and hyper-heuristics (heuristic combination rules) for bin-packing.

A large testbed is employed with 1417 instances including 1D and 2D bin packing problems (convex and non-convex) and for every instance a set of features is computed. The mapping of all instances through PCA is presented in Section 9.3.

## 9.1 Principal Component Analysis (PCA)

PCA is a useful multivariate statistical technique that has found application in fields such as face recognition and image compression. It is a common technique for finding patterns in high dimensional data [131]. The general idea behind PCA has been rediscovered and renamed several times. For example, it is called the Karhunen Loeve method in electrical engineering, empirical orthogonal functions in geophysical areas, proper orthogonal decomposition in applied mathematics, and factor analysis in many other fields [159]. Moreover, PCA is often used as a clustering technique [5, 102].

PCA identifies new variables, called the principal components, which are linear combinations of the original variables. For visualization purposes, the first two (or three) components are usually chosen as new axis for plotting all observations. However, as much information will typically be lost in two- or three-dimensional visualizations, it is important to systematically try different combinations of components. Each component can then be interpreted as the direction which maximizes the variance of the observations when projected onto the component. As the principal components are uncorrelated, they may represent different aspects of the observations. The computation of the principal components for a dataset is based on linear algebra operations. If data are standardized (with zero average and standard deviation of one unit), the principal components are normalized eigenvectors of the covariance matrix of the instances and ordered according to how much of the variation present in the data they contain.

Suppose that  $\mathbf{x}$  is a vector of  $p$  random variables. Although PCA does not ignore covariances and correlations, it concentrates on variances [84]. The main idea is to look for a linear function  $\alpha'_1 \mathbf{x}$  of the elements of  $\mathbf{x}$  having maximum variance, where  $\alpha_1$  is a vector of  $p$  constants  $\alpha_{11}, \alpha_{12}, \dots, \alpha_{1p}$  and  $'$  denotes transpose, so that

$$\alpha'_1 \mathbf{x} = \alpha_{11}x_1 + \alpha_{12}x_2 \cdots + \alpha_{1p}x_p = \sum_{j=1}^p \alpha_{1j}x_j$$

Next, look for a linear function  $\alpha'_2 \mathbf{x}$ , uncorrelated with  $\alpha'_1 \mathbf{x}$  having maximum variance, and so on, so that at the  $k$ th stage a linear function  $\alpha'_k \mathbf{x}$  is found that has maximum variance subject to being uncorrelated with  $\alpha'_1 \mathbf{x}, \alpha'_2 \mathbf{x}, \dots, \alpha'_{k-1} \mathbf{x}$ . The vector  $\alpha_1$  maximizes  $\text{var}[\alpha'_1 \mathbf{x}] = \alpha'_1 \sum \alpha_1$ , subject to  $\alpha'_1 \alpha_1 = 1$ ; where  $\sum$  is the covariance matrix. The standard approach is to use the technique of Lagrange multipliers [84]. It is shown that  $\alpha_1$  is the eigenvector corresponding to the largest eigenvalue of  $\sum$ , and  $\text{var}(\alpha'_1 \mathbf{x}) = \alpha'_1 \sum \alpha_1 = \lambda_1$ , the largest eigenvalue.

As a brief example, let us consider a 2-variable dataset plotted in Figure 9.1, in which the four larger points are special observations in some way. The vector showing the first principal component (PC1) goes through the cloud of points in the direction where the points are most spread. The second and last principal component (PC2) is orthogonal to the first. The projection of the observations over the PC1 is shown in Figure 9.1b. The closeness of the four larger points is partially preserved in this dimensionality reduction from two dimensions

to one. For 3D data, the plane through the data where the points are most spread is built by the first two principal components. This is the plane that minimizes the sum of squares of the orthogonal distances from all points to the plane. A two-dimensional visualization of the 3D data is the orthogonal projection of the points over the plane.

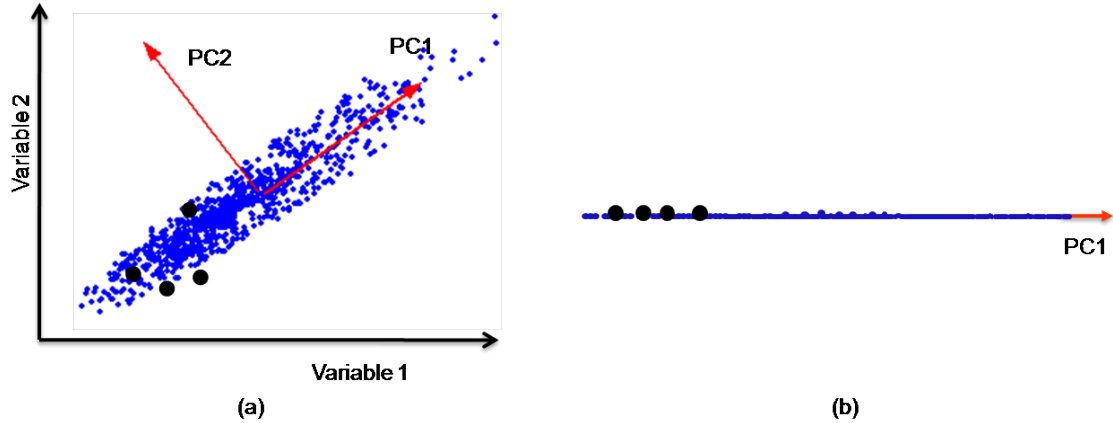


Figure 9.1: An example of principal component analysis for a dataset of 2 variables. (a) PCA identifies the two orthogonal directions (PC1 and PC2) along which the data have the largest spread. (b) Observations plotted in one dimension using their projections onto PC1.

## 9.2 Experimental Setup

This section is brief since the set of heuristics, hyper-heuristics, instances and problem features employed for the analysis were described in previous sections.

The following six selection heuristic approaches were employed:

1. First Fit Decreasing (FFD).
2. Filler.
3. Best Fit Decreasing (BFD).
4. Djang and Finch with initial fullness of 1/4 ( $DJD_{1/4}$ ).
5. Djang and Finch with initial fullness of 1/3 ( $DJD_{1/3}$ ).
6. Djang and Finch with initial fullness of 1/2 ( $DJD_{1/2}$ ).

These are the same heuristics employed in Chapter 8 and described in Section 2.6. For the 2D BPP, the heuristic Constructive Approach with Maximum Adjacency (CAD) was employed for finding the actual placement of the selected piece in a position inside the object for all the 2D instances.

Our experimental testbed comprises 1417 instances with different types and features, which are summarized in Tables 4.1 through 4.4. The *Fu* instance from Table 4.1 was not employed here since it was always solved with 2 objects by every single heuristic.

### 9.2.1 Meta-data for the Bin Packing Problem

A critical part of the proposed analysis is the identification of suitable features of the problem instances that might explain algorithm performance. Especially, the 2D BPP is a source of many possible features. Twenty three numerical features were computed for each instance (as in Section 8.1.1). The first 19 are the same features listed in Section 6.2. The last 4, related with concavities, are listed in Section 8.1.1.

In Chapter 6 we proposed a general methodology for selecting the most relevant features that can predict algorithm performance [99]. Given a set of problem features, a subset of features that are related with heuristic performance is found. This methodology was developed to select a subset of the most relevant features, but does not give information about the relationship of each feature and heuristic performance. In this chapter we attempt to find these relationships.

The methodology proposed in [99], recommends that highly correlated features are pruned as an early step, as they carry redundant information. The method also requires normalizing the algorithm performance. This is explained in Section 6.1 and computed with Equation 6.3.

In this methodology, normalized performance is then used for clustering all instances. Values for normalized performance show which heuristics are better and which heuristics are worse for a given instance and do not show how easy or hard a problem instance is. Clustering results are highly dependent on the proper choice of clustering variables [5]. So, the clusters built using normalized performance group those instances that are better solved by the same heuristics, either they are easy or hard.

When the proposed methodology was applied to all our testbed instances and the 23 features computed, nine different features related to heuristic performance were found (see Table 6.2) [100]. These nine features were chosen for the numerical representation of the evolutionary process in Chapter 8 (along with a tenth feature regarding the percentage of remaining pieces to be placed of the current instance). The same nine features are also chosen for the PCA analysis of this chapter, since it was shown that they are related with algorithm performance (the tenth feature is not relevant when dealing only with complete instances as it is the case of this particular chapter). Now, we want to find out if these nine features are related with hyper-heuristic performance. The basic question is about what feature values have those instances that are the most suitable to be solved better by the hyper-heuristics generated.

## 9.3 Results and Discussion

This section describes the main analysis done with the available data. We performed the PCA considering the 1417 instances and the 9 previously selected variables (Table 8.3) with the *R* programming language [119]. Initially, we standardized every variable (average of zero and standard deviation of 1), to ensure magnitude consistency. The first two principal components explain 42.7% and 22.3% of the variance respectively. That is, 65% of the data variation is captured by the plot in Figure 9.2a. The third principal component explains 11% of the dataset variance while the remaining 6 principal components explain jointly the remaining 24%. We select the first two principal components to plot all the dataset (Figure 9.2a). The three graphs below (Figure 9.2b.1, 9.2b.2 and 9.2b.3) show where the instances of the three main categories

(1D, convex 2D and non-convex 2D) are located. The 540 2D convex instances include 30 rectangular cases which are concentrated in the circle of Figure 9.2b.2. In these plots, close points represent instances that are similar according to the 9 variables.

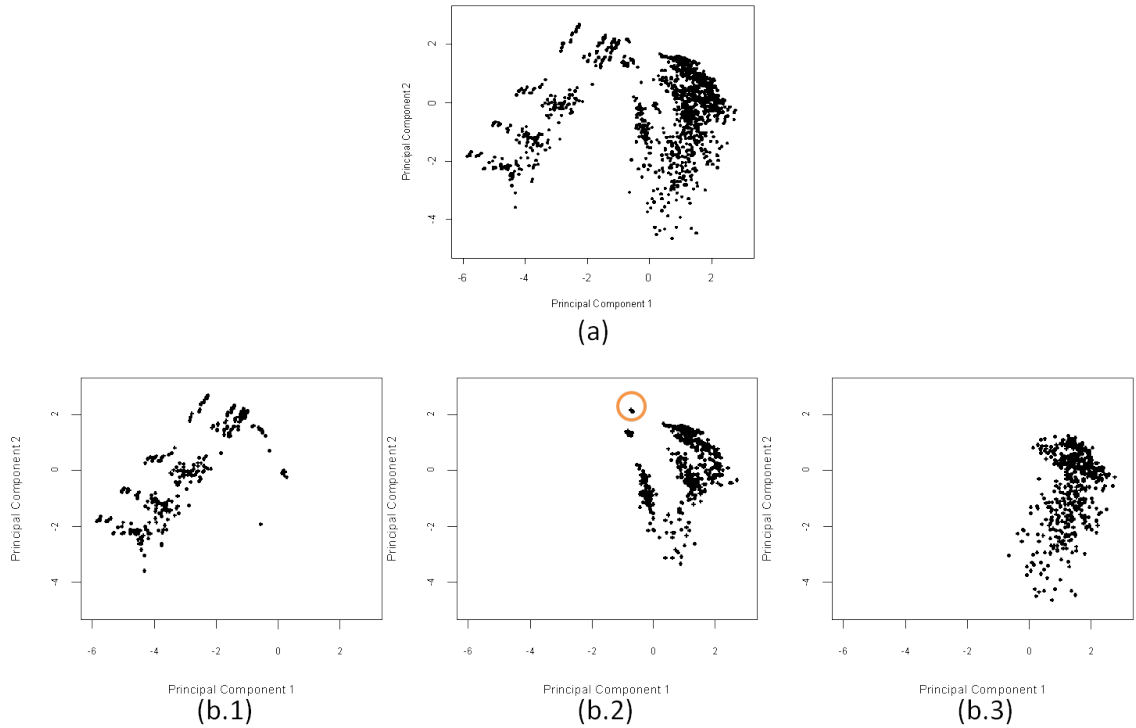


Figure 9.2: (a) The 1417 instances plotted along PC1 and PC2. (b.1) The 397 instances of the 1D BPP. (b.2) The 540 instances of the 2D BPP (convex). The 30 rectangular instances are all plotted almost in the same place inside the circle. (b.3) The 480 instances of the 2D irregular BPP (non-convex).

Each principal component is a linear combination of the 9 variables. The coefficients (called loadings) for each of the 2 main principal components are shown in Table 9.1. Usually, loadings with the largest absolute value give a meaning for the new variables PC1 and PC2.

Table 9.1: Loadings for the two main principal components of the data. Features 1 through 9 are those referred in Table 8.3. Figures with largest absolute values are in bold font.

Feature	1	2	3	4	5	6	7	8	9
	Number of pieces	Mean area	Variance of area	Mean rectangularity	Variance of rectangularity	Mean height	Variance of width	% of huge pieces	Concavity degree
PC1	<b>-0.34</b>	<b>-0.42</b>	-0.15	<b>-0.44</b>	<b>0.42</b>	0.18	<b>0.34</b>	<b>-0.36</b>	0.19
PC2	0.15	<b>-0.34</b>	<b>-0.63</b>	0.17	-0.13	-0.13	<b>-0.42</b>	<b>-0.42</b>	-0.22

When the standardized variables are multiplied by each vector of loadings we obtain the PC1 and PC2 scores, which are the horizontal and vertical coordinates for each instance

plotted on Figure 9.2. Let  $\mathbf{x}$  be the vector of the nine standardized variables for a given instance, and  $\alpha_1$  and  $\alpha_2$  be the vectors of loadings for PC1 and PC2 respectively. For example, a 2D regular instance has

$$\mathbf{x}' = [ -0.25 \quad -1.12 \quad -0.97 \quad 1.28 \quad -1.3 \quad -0.54 \quad -0.95 \quad -0.57 \quad -0.5 ]. \text{ Then,}$$

$$\mathbf{x}'\alpha_1 = [ -0.25 \quad -1.12 \quad \dots \quad -0.5 ] \begin{bmatrix} -0.34 \\ -0.42 \\ \dots \\ 0.19 \end{bmatrix} = -0.72$$

is the horizontal coordinate (PC1 score) for the given instance. The vertical coordinate is given by  $\mathbf{x}'\alpha_2$ . Loadings are plotted in Figure 9.3. These plots represent the original variables in the 2D space. Remember that the PC are obtained as linear combinations of the original variables. The loading of a single variable indicates how much this variable participates in defining the PC. Variables contributing very little to the PCs have small loading values and are plotted around the center of the plot. The nine features are distributed along the graph in the direction where

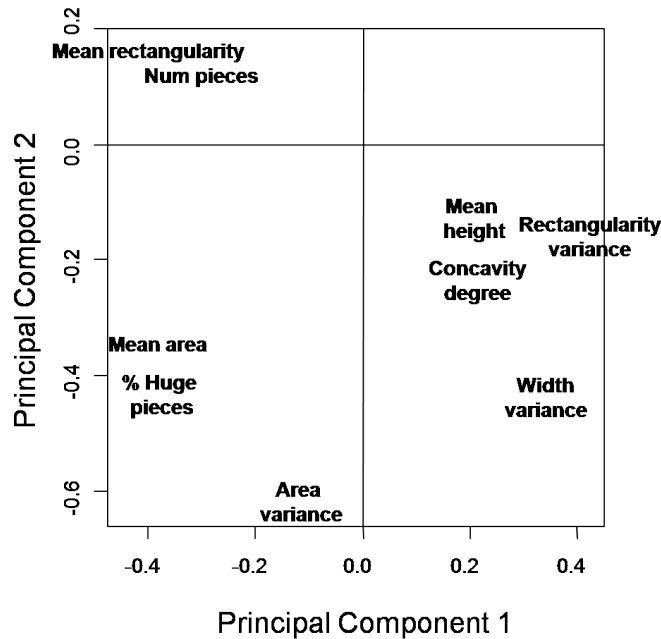


Figure 9.3: PCA 2D-loading plot of the two first principal components.

The variables *variance of rectangularity* (0.42) and *variance of width* (0.34) have a positive projection onto the first component. In consequence, the largest positive values of PC1 refer to instances with high variability of shapes and pieces width. We used the same width for all the 1D items, so 1D instances have the minimum possible value for the variance of width (zero). The *variance of width* measures variability in piece sizes only for the 2D cases. Variables *number of pieces* (-0.34), *mean area* (-0.42), *mean rectangularity* (-0.44) and *percentage of huge pieces* (-0.36) have a negative projection onto the first component. Instances with many items and large regular pieces have negative values of PC1. Therefore, PC1 separates almost perfectly the 1D instances (plotted in the left side, see Figure 9.2b.1) from the 2D instances (plotted in the right side, see Figures 9.2b.2 and 9.2b.3).



On the other principal component, PC2, the variable *variance of area* ( $-0.63$ ) has the highest negative projection. This variable measures variability in item sizes for both: 1D and 2D data. The loading is negative ( $-0.63$ ). Then, the greater the variability of size, the lower the value of PC2, which means that instances with huge variety of items sizes tend to be plotted lower in Figure 9.2a. The variables *mean area* ( $-0.34$ ), *variance of width* ( $-0.42$ ) and *percentage of huge pieces* ( $-0.42$ ) have also a negative projection onto PC2. Therefore, instances with largest items tend to be plotted lower in the graphs of Figure 9.2. All 2D regular instances have high positive values for PC2, and they are plotted in the upper part of the graph (inside the circle in Figure 9.2b.2).

### 9.3.1 Distribution of Features across the PCA Map

The distribution of the most representative input variables (features) can be visually explored using the graphs shown in Figure 9.4. Each section of the graphs can be characterized by a combination of different feature values. For example, the bottom-left part of the graph has instances with high number of pieces (black points in the bottom-left part of the first plot of Figure 9.4), high mean area, high variance of area, high mean rectangularity, low variance of rectangularity (clear points in the central plot of Figure 9.4).

### 9.3.2 Distribution of Heuristic Performance Across the PCA Map

The performance of the six single heuristics is computed with Equation 3.2 and it is mapped in each of the subplots on Figure 9.5. The pattern of gray and black points is almost the same for the 6 subplots showing that easy instances (black points) obtain high performance measure whatever the heuristic. Points in the left are 1D instances and they obtain higher measures of performance. Hardest instances are in the top-right part of the graph, where clearer points are located. These instances have the highest values for PC1 and PC2 scores. According to our analysis, the higher the PC1 and PC2 scores, the hardest the instance.

Figure 9.6 shows the *normalized* performance (computed with Equation 6.3). Black points correspond to those instances that are better solved by a particular heuristic compared to the others. Different color patterns can now be found across the 6 subplots. For example, heuristics  $DJD_{1/4}$  and  $DJD_{1/3}$  have particularly low performance for instances in the bottom-left of the map (very clear dots), when compared with the performance of the other four heuristics. This analysis suggests that those instances with low values of  $\mathbf{x}'\alpha_1$  and  $\mathbf{x}'\alpha_2$  are not especially suitable for heuristics  $DJD_{1/4}$  and  $DJD_{1/3}$ . This observation should be taken into account when designing heuristic selection techniques.

### 9.3.3 Clustering

All our testbed instances were grouped into 8 clusters according to their normalized performance of the 6 heuristics [100] (see Section 8.1.1 and Tables 8.1 and 8.2). The clustering was done with the  $k$ -means algorithm which minimizes the variance of the heuristic performance within instances in each cluster. Broadly speaking, instances with the same best heuristic and the same worst heuristic were grouped in the same cluster.

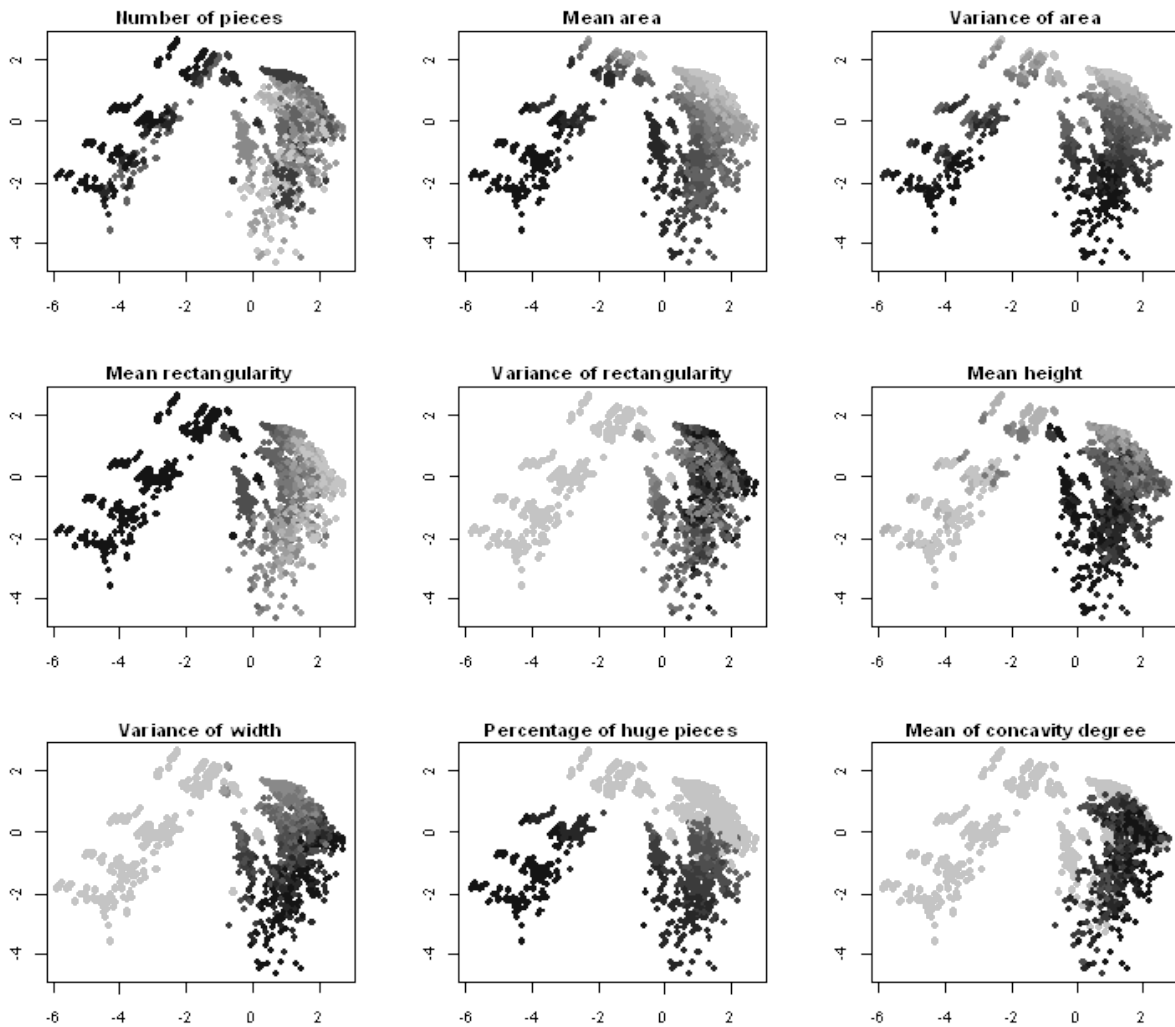


Figure 9.4: Distribution of feature values across the PCA map (black shows maximum value for each feature). Horizontal axis is for PC1 score, while vertical axis is for PC2 score.

Using the analysis in Figure 9.2, instances of the 8 clusters are plotted in Figure 9.7. The clusters were built based *only* on heuristic performance. And the principal components were computed based *only* on the 9 selected features. However, instances for most clusters seem to be somewhat concentrated for each of the subplots of Figure 9.7. For example, all of the instances plotted in the bottom-left part of Figure 9.2a belong to the same cluster (first subplot of Figure 9.7), since no one of the other 7 subplots in Figure 9.7 have points in the bottom-left part. Instances in the same cluster have approximately the same best and worst heuristics. As instances in the same cluster tend to be close in the PCA map; they also have similar features. This is a confirmation that features are indeed correlated with algorithm performance.

### 9.3.4 The Best Heuristic

If we take the number of objects as the measure of performance for the heuristics, more than one heuristic will be the best in 96% of the cases. The quality measure of Equation

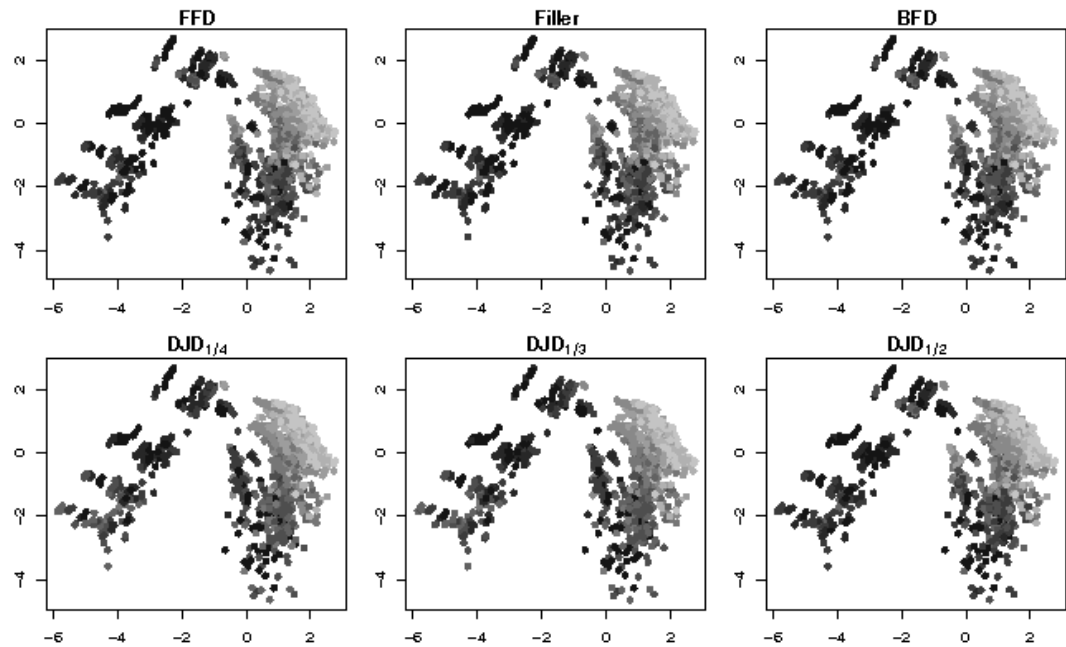


Figure 9.5: Performance of the 6 heuristics across all instances in the PCA map (black shows maximum value). Horizontal axis is for PC1 score, while vertical axis is for PC2 score.

3.2 distinguishes between solutions with the same number of objects because it rewards those solutions with filled objects or nearly so. This makes sense, because empty space concentrated in one or few objects is more likely to be useful later. That is why, Equation 3.2 is considered as a better measure of performance by some researches [143, 24, 51]. Nevertheless, with this measure several ties still occur for 58% of our 1417 instances. Preliminary analysis showed that most of these ties occur among the three 1-piece heuristics (FFD, Filler or BFD) as well as among the three DJD heuristics. Figure 9.8 superimposes a color (grey or black) in the plot from Figure 9.2a indicating whether a 1-piece heuristic or a DJD heuristic was the best. This time we discarded 16.4% of the instances because at least one of the 1-piece heuristics got the best quality measure in a tie with at least one of the DJD heuristics. 15.2% of the instances are better solved by a 1-piece heuristic, while a DJD heuristic was the best for the remaining 68.3% of the cases. This suggests that the DJD heuristics are very effective. In Figure 9.8a pattern arises, especially in the left, where grey points are concentrated (in the oval). These are 1D instances (see Figure 9.2b.1). For a given value of PC2 ( $x'_2$ ), the larger the value of PC1 ( $x'_1$ ), the more likely a 1-piece heuristic will succeed. For 1D instances, PC1 is related with the size of the items. That is, for small items 1-piece heuristics seem to be better. On the top-right part of Figure 9.8 there is also a concentration of grey points (rectangle). Higher values of PC2 are related with regular and small pieces. This section of the plot is mainly for 2D convex instances (see Figure 9.2b.2).

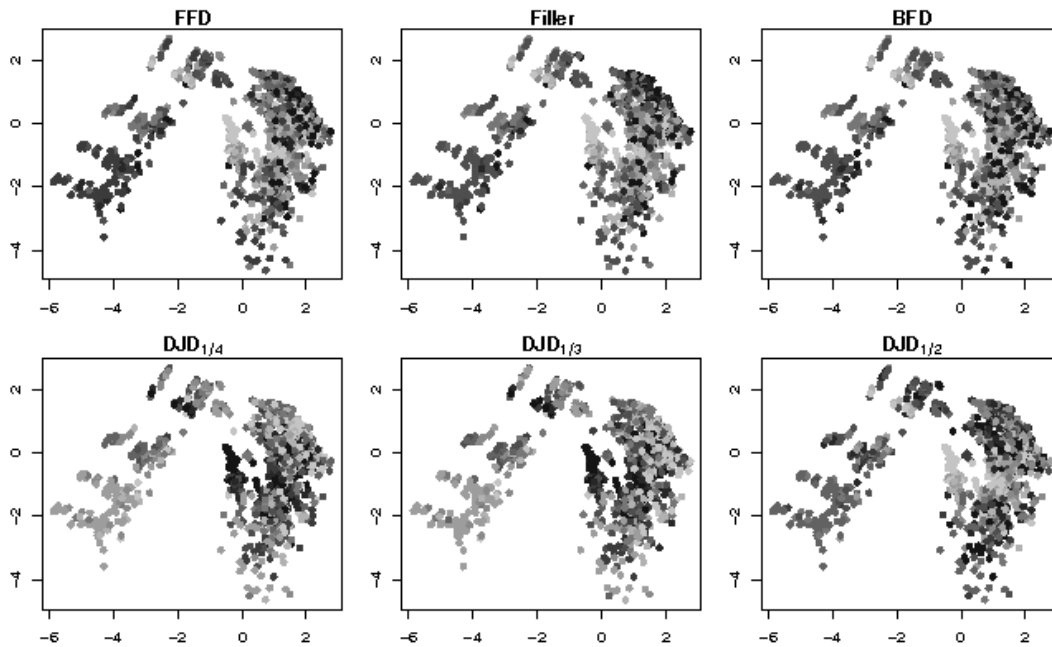


Figure 9.6: Normalized performance of the 6 heuristics across all instances in the PCA map (black shows maximum value). Horizontal axis is for PC1 score, while vertical axis is for PC2 score.

### 9.3.5 Hyper-heuristic Performance

Employing the evolutionary framework described in Chapter 8, 20 hyper-heuristics were built with different training and testing sets selected from our testbed. Each particular instance was included in the testing set of 10 different runs. Thereafter, the best hyper-heuristic for each instance is selected. Figure 9.9 marks with letters *b* and *w* those instances whose best hyper-heuristic obtained a different result (better or worse respectively) compared against the result of the best of the six heuristics. These cases are not spread along the cloud from Figure 9.2a. Rather, they are concentrated in only few sections. We are interested on which characteristics have those sections of the graph to understand which features of the BPP are able to explain hyper-heuristic performance. Moreover, the best and worst cases are mixed in these sections, which means that this particular PCA analysis is able to show which instances are likely to be solved *different* by the hyper-heuristic compared with the best single heuristic, but does not distinguish between solving cases with fewer or more objects. The concentration of fifteen *b*'s and three *w*'s in the rectangle of Figure 9.9 corresponds only to 1D instances and they have the higher values of PC1 and PC2 for 1D instances (see Figure 9.2b.1). For those loadings in Table 9.1 that regards 1D, *mean area* (-0.42) has the highest impact in PC1 and *variance of area* (-0.63) in PC2. Both loadings are negative. So, 1D instances with small items with similar sizes characterize the zone in the rectangle of Figure 9.9. Instances here are likely to be solved with a different number of bins by hyper-heuristics than by the best heuristic. In

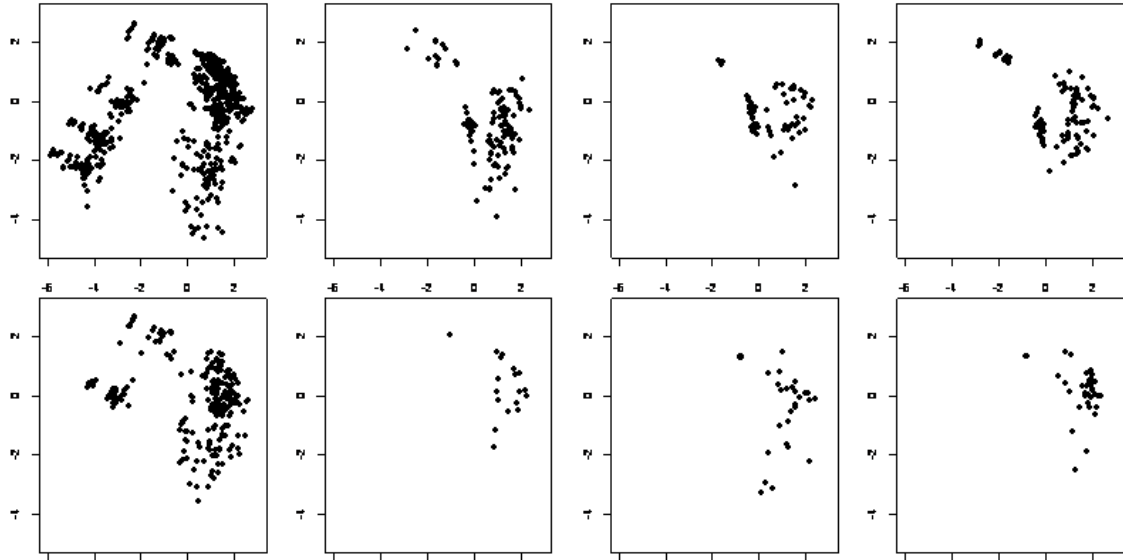


Figure 9.7: Instances in each of the 8 clusters built based on the normalized performance. Horizontal axis is for PC1 score, while vertical axis is for PC2 score.

fact, 10 of the  $b$ 's in the rectangle represent instances with 501 items whose optimum has 3 items per bin (type *Trip501* in Tables 4.1 and 4.2). Which means that hyper-heuristics found a better result for half of the *Trip501* instances than any of the 6 heuristics. The other 1D cases in the rectangle are instances of *DB2* and *Trip249*, all with smaller items than the 1D average. The  $b$ 's and  $w$ 's outside the rectangle are for convex and non-convex instances. Most of them have PC2 near 0 (inside the circle). That is, we have  $\mathbf{x}'\alpha_2 \approx 0$  in many 2D instances for which the hyper-heuristics produce different results than the best single heuristics. We have found that the particular combination of features given by  $\mathbf{x}'\alpha_2$  has an effect in hyper-heuristic performance. Maybe, other hyper-heuristic methods should be tried for those 2D instances that do not hold  $-\text{threshold} < \mathbf{x}'\alpha_2 < \text{threshold}$  to try to find different results than the single heuristics.

For some instances, hyper-heuristics achieve better results than the best single heuristic for that instance. These cases support the use of hyper-heuristics, since for some applications, any reduction in material is extremely valuable. For most instances, the evolved hyper-heuristic produces the same quality than the best single heuristic. This is also beneficial, however, as the choice of best heuristic varies from instance to instance. Using a hyper-heuristic may be, therefore, preferable than selecting a single heuristic for all problem instances [100]. After the hyper-heuristic is evolved, the computational cost of applying a generated hyper-heuristic to a problem is lower than the time used in solving with all heuristics and then choosing the best result [143].

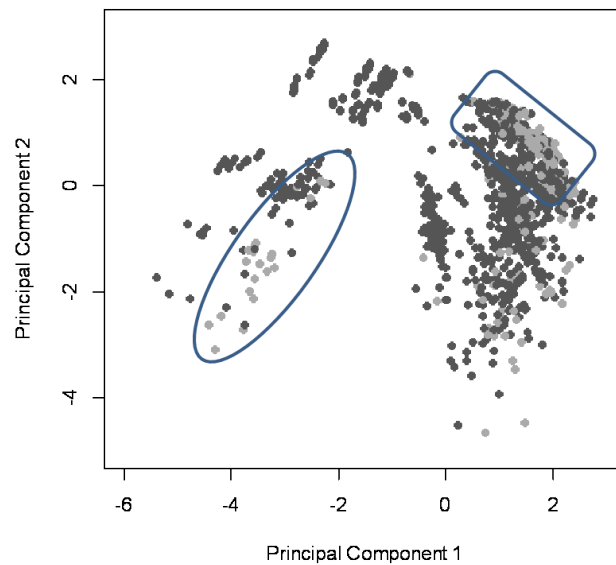


Figure 9.8: Best heuristic. Color grey is for instances better solved by 1-piece heuristics (FFD, Filler or BFD). Points in color black are instances better solved by the DJD heuristics.

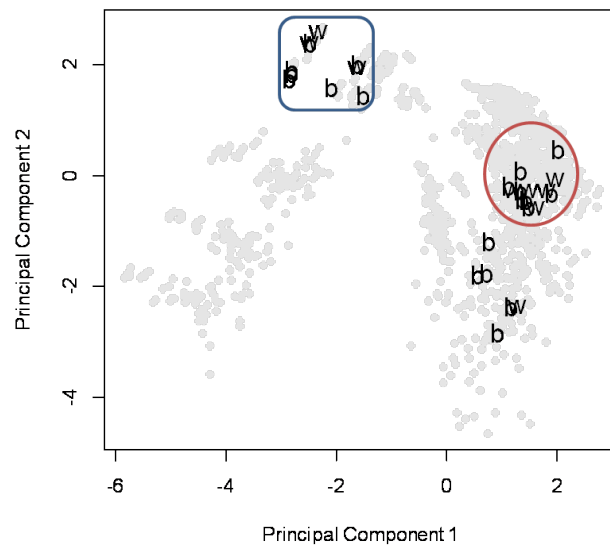


Figure 9.9: From all instances, only 28 were solved with fewer objects by the best hyper-heuristic than any of the 6 heuristics (plotted with the letter **b**). Whilst in 9 cases, none hyper-heuristic could reach the best single heuristic result (plotted with the letter **w**). In all the remaining cases, the best hyper-heuristic got the same number of objects than the best heuristic for each case (gray dots).

## 9.4 Summary

The analysis presented in this chapter constitutes a first step into the multivariate nature of the BPP characterization. Understanding why some instances are more suitable for some heuristics and hyper-heuristics is well worth the effort. Therefore, we expect this understanding will help in the design of better solution approaches in the future.

We have considered a large dataset which makes the analysis reliable and robust. The dataset contains a wide range of feature values. We found that PCA can help us to characterize the BPP and relate some feature combinations with algorithm performance. For example, heuristics  $DJD_{1/4}$  and  $DJD_{1/3}$ , although very effective, have a particularly unsuitability for instances with low values of both  $\mathbf{x}'\alpha_1$  and  $\mathbf{x}'\alpha_2$  (where  $\mathbf{x}$  is the vector of standardized features and  $\alpha_1$  and  $\alpha_2$  are the loading vectors from Table 9.1). The BPP has a complex structure. There are not simple rules relating features and algorithm performance. It may be necessary to consider feature combinations and interaction among features in order to have a clearer insight into performance prediction. This contrasts with other combinatorial optimization problems. For example, in the constraint satisfaction problem, a couple of well selected features (density and tightness) are enough to predict which of two heuristics will be the best [110].





# Chapter 10

## Final Conclusions and Future Work

This chapter summarizes what has been presented throughout this dissertation document, emphasizing on the most important details, the main contributions and the direction that this research would take in the future of intelligent algorithm selection.

### 10.1 Dissertation Summary and General Discussion

An evolutionary framework that produces hyper-heuristics for solving the Bin Packing Problem is proposed. A basic implementation of the solution model is performed for the 2D Irregular BPP (convex). Afterwards, the main elements of the framework are analyzed and improved. A following implementation of the model was conducted at the same time that the range of tackled instances is increased to include concave shapes and 1D instances. Finally, a knowledge discovery approach is employed to find insights into the BPP structure.

#### 10.1.1 Hyper-heuristics for Solving 2D Irregular Bin Packing Problems (convex)

A first implementation of the solution model is applied for the 2D irregular BPP that includes only convex polygons. Averaging all GA runs, in 88.6% of the instances, the hyper-heuristic found the same result than the best single heuristic (average from Tables 5.2 to 5.5). Taking into account that the best single heuristic may be different for each instance, this result is an indication that the hyper-heuristic is able to learn and select the best heuristic for most of the cases. In one run, the hyper-heuristic beats the best heuristic in 0.4% of the cases (Table 5.3). Besides, the average hyper-heuristic gets 1.44 fewer objects against the average single heuristic (Table 5.6).

After this model was run and tested, several changes in the main elements of the framework were proposed.

#### 10.1.2 Defining a Problem-State Representation Scheme

A methodology was proposed for selecting the most meaningful features to comprised the state representation. The general idea for identifying those features related with heuristics

performance is based on clustering instances according to heuristic performance and then features able to predict with cluster membership are selected.

We replicated experiments from the first implementation of the model (Chapter 5, Section 10.1.1), with all elements equal (single heuristic repository, instances testbed), except for the problem state representation scheme. Some GA parameters were adjusted before replicating experiments with both representations.

Averaging the four experiments, we get the following results for the training instances: In 35.1% of the instances, hyper-heuristics built with the new representation scheme performed better than those evolved with the previous representation scheme. In 65.6% of the instances both representations performed the same, while in the remaining 3.3% of the instances the new hyper-heuristics could not reach the results gotten by the previous hyper-heuristics. For the testing instances, the percentages are 25.5%, 68.5% and 6.0% for better, equal and worse performance respectively, when employing the representation scheme built with the proposed methodology (average from Table 6.3). Another interesting finding is about the population average fitness of the GA runs. Employing the new representation scheme, the fitness of the whole population was higher (Figure 6.6).

### **10.1.3 The DJD Heuristic as an Effective Heuristic for the 2D Irregular BPP**

The most effective heuristic employed in the first implementation of the model was studied deeper and a time-saving routing was implemented. In the original implementation, several unnecessary computations were performed trying to place the same piece in the same object several times. These cases occur when the piece was paired with different other pieces. The main idea was to keep in memory those results when a piece was already tried to be placed in a given object with exactly the same placed pieces. There is an average computational time reduction of 80% when compared to the case where no record is kept at all. We also explored different initial levels of fullness before trying to place combinations of pieces within an allowed waste, namely,  $1/4$ ,  $1/3$ ,  $1/2$  and  $2/3$ . Initial fullness of  $1/4$  and  $1/3$  variations are the most effective and there is no significant difference between both (Figure 7.2).

### **10.1.4 Hyper-heuristics for Solving 1D and 2D Irregular Bin Packing Problems (concave)**

In this second implementation of the model, we had a reviewed heuristic repository and a more robust way for defining the representation scheme. Besides, we also reviewed the GA parameters. But the main change performed was the capability of the model to handle concave instances as well as 1D instances. That is why, we claim that this is a more general evolutionary framework. Results from the hyper-heuristics generated are compared against the best single heuristic for each case. Averaging the four experiments conducted for the testing instances, in 91.3% of the instances, the average hyper-heuristic gets the same result than the best single heuristic (average from Table 8.5). This number can be compared with the 88.6% from the first implementation (Section 10.1.1). In one run, the average hyper-heuristic beats the best heuristic in 0.6% of the cases for this run.

### 10.1.5 A Knowledge Discovery Approach for Understanding how Features Impact in Heuristic and Hyper-heuristic performance

Principal Component Analysis was employed to plot in 2D all our 1417 testbed instances. With this dimensionality reduction technique, it was possible to build two variables that retain about  $2/3$  of the variation of the data originally defined with 9 variables. The variables chosen to characterize the BPP were the 9 features selected for the representation scheme in Section 8.5, since it was shown that they are indeed related with heuristic performance. We found some interesting relations between features and heuristic performance. For example, some sections of the plot, characterized by particular combinations of features, are particularly unsuited to be solved by heuristics  $DJD_{1/3}$  and  $DJD_{1/4}$  (Figure 9.6), even though these two heuristics are the best when compared with 9 other selection heuristics (Figure 7.2).

### 10.1.6 General Discussion

Terashima-Marín et al. [138] proposed in 2006 an evolutionary model that produces hyper-heuristics for the 2D regular BPP. This solution model got promising results. Therefore, it was generalized to the 2D irregular BPP in this dissertation. We obtained encouraging initial results at the same time that some improvements were proposed and successfully tested. Thereby, this investigation contributes with an evolutionary framework that produces good quality hyper-heuristics for solving several types of Bin Packing Problems: 1D and regular and irregular 2D (including convex and concave polygons). Besides increasing the level of generality of the model in terms of the kind of instances handled, the model is now more robust, since the most relevant features are chosen for state representation and the quality of the instance repository was improved. We hypothesize that these improvements are the reasons why the GA process converges faster (comparing the 500 generations of a 100-individual population from our first implementation of the model (Chapter 5) with the 80 generations of a 30-individual population from the most recent implementation of the model (Chapter 8)).

The most common result among the hyper-heuristics produced and tested is the same than the best single heuristic result per case. This is an indication that the hyper-heuristics can learn which is the best heuristic for each instance. For some cases the hyper-heuristic even obtains a solution that employs fewer objects than the best single heuristic. For real world applications these cases justify the existence of the hyper-heuristic since a small reduction of material could be translated into large savings of economical costs, especially when the layout is to be performed many times, as in the shoe and garment industry. Unfortunately, the hyper-heuristics produced could not reach the best heuristic result in few cases.

Each of the hyper-heuristics produced can be considered as other possible heuristic. If there is enough time for finding a solution, the general recommendation is to solve the instance at hand with every single heuristic plus one or several hyper-heuristics. Then choose the best result. The time required to solve any instance from our testbed goes from a few seconds to a couple of minutes. Hyper-heuristics do not employ more time than the most effective single heuristics (Table 8.16). Therefore, if time is limited and we need to choose only one heuristic to find a solution, a good decision is to employ a hyper-heuristic to do so. Heuristics and hyper-heuristics considered are single pass (never revise a piece already placed) which makes them faster than other solution algorithms. In conclusion, the proposed hyper-heuristic

generation framework is to be used mainly when time is a constraint.

## 10.2 Main Contributions

The overall main contribution of this dissertation is the building of an evolutionary framework suitable to combine heuristics for solving several types of BPP. One of the main advantages of the developed framework is that this hyper-heuristic approach is problem independent and can be easily utilized by non-experts as well. By developing this framework, the following contributions were developed.

### 10.2.1 Hyper-heuristics for Solving 2D Irregular Bin Packing Problems (convex)

As far as we know, this is the first heuristic-combination approach for solving the 2D irregular BPP. Moreover, there are so few studies about the 2D irregular BPP, so this dissertation presents one of the first solution approaches for the irregular version of the 2D BPP. Some of the heuristics presented were adapted from the regular (rectangular) version or from other versions of the C&P problem (such as the Strip Packing Problem). An algorithm for randomly generating 2D irregular instances with convex polygons was designed and programmed.

### 10.2.2 Defining a Problem-State Representation Scheme

We present a new way of devising a representation for problem states. The proposed methodology can be applied to any solution approach that requires summarize instance states through numerical vectors, such as Case-based reasoning. The proposed robust methodology for feature selection does not require much knowledge domain. Besides, the notion of normalized performance was first introduced in this research to measure the suitability of single heuristics to solve particular instances.

### 10.2.3 The DJD Heuristic as an Effective Heuristic for the 2D Irregular BPP

The definition of the algorithm for the 2D version of the DJD heuristics leads to effective solutions with less computational cost. We also conducted an analysis of two important parameters of the DJD heuristic: the initial level of fullness and the allowed waste incremental; finding those values where the DJD heuristic gets the best results.

### 10.2.4 Hyper-heuristics for Solving 1D and 2D Irregular Bin Packing Problems (concave)

A framework that produces rules for combining single heuristics was proposed for solving 1D BPP instances as well as 2D regular and irregular BPP cases. Some new measures for irregularity in convex and concave pieces were defined for this dissertation. As no benchmark

instances were available for testing the framework, an algorithm for randomly generating instances with concave polygons was designed and programmed. A review and improvement of the parameters employed in the genetic algorithm of the solution model was performed as well.

### **10.2.5 A Knowledge Discovery Approach for Understanding how Features Impact in Heuristic and Hyper-heuristic performance**

A methodology for exploring features and heuristic performance was proposed and applied for the BPP. As far as we know, this is the first time that the Principal Component Analysis technique is used for analyzing the BPP structure. We found some feature value combinations where the hyper-heuristics get different results than the best single heuristics.

## **10.3 Future Direction**

Some ideas for further research are presented here:

- We are interested in testing the proposed methodology for feature selection (Chapter 6) in other problems and approaches in which a characterization of instances is needed. For example, the case-based reasoning approach applied to timetabling problems [33]. When doing this, other clustering techniques may be tried as well as other models for predicting a nominal variable based on several numerical variables (features).
- In this dissertation, all features employed in instance representation were related to the pieces to be placed. But, one or several features of the instance state representation could actually describe the state of the open objects. This is because the suitability of one heuristic for solving an instance state may depend not only on the remaining pieces, but also, on the state of the objects partially filled. For example, in a new instance to be solved, all objects are empty; in contrast with an instance with few pieces remaining where several objects may have some free areas with different sizes and shapes. As far as we know, none of the works which have dealt with numerical representation of instances states for bin packing [128, 138, 139, 143] have considered this issue. For example, one numerical term in the representation vector could refer to the number of open objects available to choose from, or to the open objects percent of free area.
- The adaptation of the DJD heuristic to the 2D Bin Packing Problem was tested in regular and convex instances (Chapter 7). It would be interesting to analyze the DJD heuristic performance in instances with concave polygons.
- On one hand, many of the hyper-heuristics produced by our framework employ the same single heuristic to solve all instances. Maybe because hyper-heuristics contain the same action in all their blocks or maybe because some actions are not reachable by the states of the testing instances. On the other hand, those hyper-heuristics that beat the performance of all single heuristics are those which combine several single heuristics in the solution process. That is why it is a good idea to try penalizing the fitness of a chromosome that uses only one heuristic, by some small amount.

- When solving testing instances, the same single heuristic is applied to place many pieces before changing to another heuristic. For future work, we recommend trying the same single heuristic several times before recomputing the instance state for choosing another hyper-heuristic block. This will reduce computational time.
- Some other methodological elements can be introduced to further testing of the solution model; such as,  $k$ -fold validation and comparison against random generated hyper-heuristics.
- The proposed framework can be tested when solving the BPP with some common constraints; for example, the requirement of guillotinable cuts, different rotation allowed for different shapes or object stocks of different material quality.
- In the future, we would like to focus on characterizing the BPP with different sets of features.
- The analysis in Chapter 9 was done employing a large set of instances from different types of BPP. Extra analysis considering these types of BPP separately may lead to some interesting findings.
- Other ways for mapping in 2D data characterized by a larger number of features may be explored in the future. For example, the Self-organizing maps [90].
- We would like to develop a hyper-heuristic solution approach based on selecting a single heuristic for a given instance according to its place in the PCA map produced in Chapter 9. The idea is to develop a hyper-heuristic represented by a matrix [111].
- The PCA analysis for finding relations among features and heuristic performance could be applied to other optimization problems.

## 10.4 Closing Remarks

It is advisable that future work related to hyper-heuristic search takes into account these closing remarks.

It is important to carefully choose the set of features to describe the problem structure. They have to be able to characterize instance and problem structure as well as differentiate algorithm performance [135]. Furthermore, features must avoid redundancy and show different aspects of the problem structure. For example, in the Bin Packing Problem, it is possible to have different features related to the size of the items, such as average item area, percentage of small items, etc. We must expect some level of independence among the different features considered.

Finally, a brief note about the use of the term hyper-heuristic. A different perspective to the framework proposed is the following. We could have named *hyper-heuristic* the evolutionary process, since it can be considered as a heuristic that searches a space of heuristic combinations. Then, every chromosome would be called a *generated* heuristic able to solve the problem in a deterministic way. So, in this case we would be dealing with a hyper-heuristic of the kind of *heuristic generation* instead of the kind of *heuristic selection*.

# Appendix A

## Contributed Scientific Publications

List of scientific publications that the conducted research produced.

- LÓPEZ-CAMACHO, E., TERASHIMA-MARÍN, H., ROSS, P. AND OCHOA, G. Evolving General Heuristic Combination Rules for Various Types of Bin Packing Problems (To be submitted).
- LÓPEZ-CAMACHO, E., TERASHIMA-MARÍN, H. AND OCHOA, G. An Effective Heuristic for the Two-dimensional Irregular Bin Packing Problem *Annals of Operations Research* (Under review).
- LÓPEZ-CAMACHO, E., TERASHIMA-MARÍN, H., OCHOA, G. AND CONANT-PABLOS, S. E. Towards a deeper understanding of the bin packing problem structure *International Journal of Production Economics. Special Issue in Cutting and Packing* (Under review).
- LÓPEZ-CAMACHO, E., TERASHIMA-MARÍN, H. AND CONANT-PABLOS, S. E. The impact of the bin packing problem structure in hyper-heuristic performance. In *GECCO '12: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation* (2012) (to appear).
- LÓPEZ-CAMACHO, E., TERASHIMA-MARÍN, H. AND ROSS, P. A hyper-heuristic for solving one and two-dimensional bin packing problems. In *GECCO '11: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (2011), Natalio Krasnogor (Ed.), ACM, New York, NY, USA, pp. 257–258. DOI=10.1145/2001858.2002003
- LÓPEZ-CAMACHO, E., TERASHIMA-MARÍN, H., AND ROSS, P. Defining a problem-state representation with data mining within a hyper-heuristic model which solves 2D irregular bin packing problems. In *Advances in Artificial Intelligence IBERAMIA* (2010), Á. F. Kuri-Morales and G. R. Simari, Eds., vol. 6433 of *Lecture Notes in Computer Science*, Springer, pp. 204–213. *Best Student Paper Award*.

- LÓPEZ-CAMACHO, E., TERASHIMA-MARÍN, H., ROSS, P. AND VALENZUELA-REN-DÓN, M. Problem-state representations in a hyper-heuristic approach for the 2D irregular BPP. In *GECCO '10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation* (2010), ACM, New York, NY, USA, pp. 297–298. DOI=10.1145/1830483.1830539
- TERASHIMA-MARÍN, H., ROSS, P., FARÍAS ZÁRATE, C. J., LÓPEZ-CAMACHO, E., AND VALENZUELA-REN-DÓN, M. Generalized hyper-heuristics for solving 2D regular and irregular packing problems. *Annals of Operations Research* 179 (2010), 369-392. DOI=10.1007/s10479-008-0475-2



# Bibliography

- [1] ADAMOWICZ, M. The optimal two-dimensional allocation of irregular multiply-connected shapes with linear, logical and geometric constraints. Tech. Rep. AD0703723, New York Univ. Bronx Lab. for Electrosience research, 1979.
- [2] ADAMOWICZ, M., AND ALBANO, A. Nesting two-dimensional shapes in rectangular modules. *Computer-Aided Design* 8, 1 (January 1976), 27–33.
- [3] ALLEN, S. D., BURKE, E. K., AND KENDALL, G. A hybrid placement strategy for the three-dimensional strip packing problem. *European Journal of Operational Research* 209, 3 (2011), 219–227.
- [4] ANAND, S., MCCORD, C., SHARMA, R., AND BALACHANDER, T. An integrated machine vision based system for solving the nonconvex cutting stock problem using genetic algorithms. *Journal of Manufacturing Systems* 18, 6 (1999), 396–415.
- [5] ANZANELLO, M. J., AND FOGLIATTO, F. S. Selecting the best clustering variables for grouping mass-customized products involving workers’ learning. *International Journal of Production Economics* 130 (January 2011), 268–276.
- [6] ARAYA, I., NEVEU, B., AND RIFF, M.-C. An efficient hyperheuristic for strip-packing problems. In *Adaptive and Multilevel Metaheuristics*, C. Cotta, M. Sevaux, and K. Srensen, Eds., vol. 136 of *Studies in Computational Intelligence*. Springer, 2008, pp. 61–76.
- [7] ART, R. C. An approach to the two dimensional irregular cutting stock problem. *IBM Cambridge Scientific Centre, Report 36-Y08*. (1966).
- [8] ARTHUR, D., AND VASSILVITSKII, S. How slow is the k-means method? In *SCG ’06: Proceedings of the twenty-second annual Symposium on Computational Geometry* (New York, NY, USA, 2006), ACM, pp. 144–153.
- [9] BAASE, S., AND GELDER, A. V. *Algoritmos Computacionales, Introducción al análisis y diseño*. Addison Wesley, tercera edición, 2000.
- [10] BENNELL, J. A., AND DOWSLAND, K. A. Hybridising tabu search with optimisation techniques for irregular stock cutting. *Management Science* 47, 8 (August 2001), 1160–1172.

- [11] BENNELL, J. A., AND OLIVEIRA, J. F. The geometry of nesting problems: A tutorial. *European Journal of Operational Research* 184, 2 (January 2008), 397–415.
- [12] BENNELL, J. A., AND OLIVEIRA, J. F. A tutorial in irregular shape packing problems. *Journal of the Operational Research Society* 60, S1 (February 2009), S93–S105.
- [13] BENNELL, J. A., AND SONG, X. A comprehensive and robust procedure for obtaining the nofit polygon using minkowski sums. *Computers & Operations Research* 35 (January 2008), 267–281.
- [14] BENNELL, J. A., AND SONG, X. A beam search implementation for the irregular shape packing problem. *Journal of Heuristics* 16, 2 (2010), 167–188.
- [15] BILGIN, B., ÖZCAN, E., AND KORKMAZ, E. E. An experimental study on hyper-heuristics and exam timetabling. In *Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling* (2006), pp. 123–140.
- [16] BIRGIN, E. G., LOBATO, R. D., AND MORABITO, R. An effective recursive partitioning approach for the packing of identical rectangles in a rectangle. *Journal of the Operational Research Society* (2009).
- [17] BLAZEWICZ, J., DROZDOWSKI, M., SONIEWICKI, B., AND WALKOWIAK, R. Two-dimensional cutting problem - basic complexity results and algorithms for irregular shapes, 1989.
- [18] BOUNSAYTHIP, C., AND MAOUCHE, S. Irregular shape nesting and placing with evolutionary approach. In *IEEE International Conference on Systems, Man and Cybernetics* (1997), vol. 4, pp. 3425–3430.
- [19] BROOKS, R., SMITH, C. A. B., STONE, A., AND TUTTE, W. T. The dissection of rectangles into squares. *Duke Mathematical Journal* 7 (1940), 312–340.
- [20] BURKE, E. K., CURTOIS, T., HYDE, M. R., KENDALL, G., OCHOA, G., PETROVIC, S., RODRÍGUEZ, J. A. V., AND GENDREAU, M. Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In *IEEE Congress on Evolutionary Computation* (2010), pp. 1–8.
- [21] BURKE, E. K., CURTOIS, T., HYDE, M. R., OCHOA, G., AND VÁZQUEZ RODRÍGUEZ, J. A. HyFlex: A Benchmark Framework for Cross-domain Heuristic Search. *ArXiv e-prints* (July 2011).
- [22] BURKE, E. K., HART, E., KENDALL, G., NEWALL, J., ROSS, P., AND SCHULENBURG, S. Hyper-heuristics: An emerging direction in modern research technology. In *Handbook of Metaheuristics* (2003), Kluwer Academic Publishers, pp. 457–474.
- [23] BURKE, E. K., HELLIER, R. S. R., KENDALL, G., AND WHITWELL, G. A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem. *Operations Research* 54, 3 (2006), 587–601.

- [24] BURKE, E. K., HELLIER, R. S. R., KENDALL, G., AND WHITWELL, G. Complete and robust no-fit polygon generation for the irregular stock cutting problem. *European Journal of Operational Research* 179, 1 (May 2007), 27–49.
- [25] BURKE, E. K., HYDE, M. R., KENDALL, G., OCHOA, G., ÖZCAN, E., AND WOODWARD, J. Exploring hyper-heuristic methodologies with genetic programming. In *Collaborative Computational Intelligence*. Springer-Verlag, 2009.
- [26] BURKE, E. K., HYDE, M. R., KENDALL, G., OCHOA, G., ÖZCAN, E., AND WOODWARD, J. *A Classification of Hyper-heuristic Approaches*, vol. 146 of *International Series in Operations Research & Management Science*. Springer US, 2010, pp. 449–468.
- [27] BURKE, E. K., HYDE, M. R., KENDALL, G., AND WOODWARD, J. A genetic programming hyper-heuristic approach for evolving two dimensional strip packing heuristics. Tech. Rep. NOTTCS-TR-2008-2, School of Computer Science and Information Technology. University of Nottingham., 2008.
- [28] BURKE, E. K., AND KENDALL, G. Applying ant algorithms and the no fit polygon to the nesting problem. In *Australian Joint Conference on Artificial Intelligence* (London, UK, 1999), Springer-Verlag, pp. 453–464.
- [29] BURKE, E. K., AND KENDALL, G. Applying simulated annealing and the no fit polygon to the nesting problem. In *Proceedings of the World Manufacturing Congress* (1999), pp. 27–30.
- [30] BURKE, E. K., AND KENDALL, G. Implementation and performance improvement of the evaluation of a two dimensional bin packing problem using the no fit polygon. Tech. rep., University of Nottingham, 1999. Report ASAP99001.
- [31] BURKE, E. K., KENDALL, G., SOUBEIGA, E., COSTA, E., MARÍN-BLÁZQUEZ, J., AND ROSS, P. Constructive and local-search based hyperheuristics: A case for hybridisation?
- [32] BURKE, E. K., LI, J., AND QU, R. Data mining: an aid towards more efficient hyper-heuristic search. In *7th International Conference on the Practice and Theory of Automated Timetabling (PATAT2008)* (2008).
- [33] BURKE, E. K., PETROVIC, S., AND QU, R. Case-based heuristic selection for timetabling problems. *Journal of Scheduling* 9, 2 (2006), 115–132.
- [34] BURKE, E. K., WOODWARD, J., HYDE, M. R., AND KENDALL, G. Automatic heuristic generation with genetic programming: Evolving a jack-of-alltrades or a master of one. In *Genetic and Evolutionary Computation Conference, GECCO'07* (2007), pp. 7–11.
- [35] BUTZ, M., AND WILSON, S. An algorithmic description of XCS. In *Advances in Learning Classifier Systems*, P. Luca Lanzi, W. Stolzmann, and S. Wilson, Eds.,

- vol. 1996 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2001, pp. 267–274.
- [36] CARNIERI, C., AND MENDOZA, G. A. A fractional algorithm for optimal cutting of lumber into dimension parts. *Annals of Operations Research* 95 (2000), 83–92.
- [37] CHEESEMAN, P., KANEFSKY, B., AND TAYLOR, W. M. Where the really hard problems are. In *Proceedings of the 12th International Joint Conferences on Artificial Intelligence (IJCAI)* (Sidney, Australia, 1991), pp. 331–337.
- [38] CHIANG, M., AND MIRKIN, B. Experiments for the number of clusters in k-means. In *Progress in Artificial Intelligence*, J. Neves, M. F. Santos, and J. M. Machado, Eds., vol. 4874 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, ch. 33, pp. 395–405.
- [39] COWLING, P. I., KENDALL, G., AND SOUBEIGA, E. A hyperheuristic approach to scheduling a sales summit. In *PATAT '00: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III* (London, UK, 2001), vol. 2079, Springer-Verlag, pp. 176–190.
- [40] CRISPIN, A. J., CLAY, P., TAYLOR, G. E., BAYES, T., AND REEDMAN, D. Genetic algorithms applied to leather lay plan material utilization. In *Proceedings of the Institution of Mechanical Engineers* (2003), vol. 217, 12, ProQuest Science Journals, p. 1753.
- [41] CRUZ-REYES, L., GÓMEZ-SANTILLÁN, C., PÉREZ-ORTEGA, J., LANDERO, V., QUIROZ, M., AND OCHOA, A. Algorithm selection: From meta-learning to hyperheuristics. In *Intelligent Systems*, V. M. Koleshko, Ed. InTech, 2012, ch. 4, pp. 77–102.
- [42] CUESTA-CANADA, A., GARRIDO, L., AND TERASHIMA-MARÍN, H. Building hyper-heuristics through ant colony optimization for the 2D bin packing problem. In *KES (4)* (2005), R. Khosla, R. J. Howlett, and L. C. Jain, Eds., vol. 3684 of *Lecture Notes in Computer Science*, Springer, pp. 654–660.
- [43] DEB, K., AND GOLDBERG, D. E. mGA in C: A messy genetic algorithm in C, 1991.
- [44] DOWSLAND, K. A., AND DOWSLAND, W. B. Solution approaches to irregular nesting problems. *European Journal of Operational Research* 84 (1995), 506–521.
- [45] DOWSLAND, K. A., DOWSLAND, W. B., AND BENNELL, J. A. Jostling for position: local improvement for irregular cutting patterns. *Journal of the Operational Research Society* 49, 6 (1998), 647–658.
- [46] DOWSLAND, K. A., VAID, S., AND DOWSLAND, W. B. An algorithm for polygon placement using a bottom-left strategy. *European Journal of Operational Research* 141, 2 (September 2002), 371–381.
- [47] DUCATELLE, F., AND LEVINE, J. Ant colony optimisation for bin packing and cutting stock problems. In *In Proceedings of the UK Workshop on Computational Intelligence* (2001).

- [48] DYCHOFF, H. A typology of cutting and packing problems. *European Journal of Operational Research* 44 (1990), 145–159.
- [49] ELMAGHRABY, A. S., ABDELHAFIZ, E., AND HASSAN, M. F. An intelligent approach to stock cutting optimization. In *Proceedings of ISCA 13th international conference on computer applications in industry and engineering (CAINE-2000)* (USA, 2000), pp. 90–93.
- [50] FALKENAUER, E. A Hybrid Grouping Genetic Algorithm for Bin Packing. *Journal of Heuristics* 2, 1 (1996), 5–30.
- [51] FALKENAUER, E., AND DELCHAMBRE, A. A genetic algorithm for bin packing and line balancing. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation* (1992), pp. 1186–1192.
- [52] FARÍAS-ZÁRATE, C. J. Hiperheurísticas mediante un algoritmo genético con longitud variable para resolver problemas de corte de material en dos dimensiones. Master's thesis, Tecnológico de Monterrey, May 2006. Advisor: Dr. Hugo Terashima-Marín.
- [53] FEKETE, S. P., KÓHLER, E., AND TEICH, J. Higher-dimensional packing with order constraints. *SIAM Journal on Discrete Mathematics* (2006), 1056–1078.
- [54] FUJITA, K., AKAGI, S., AND KIROKAWA, N. Hybrid approach for optimal nesting using a genetic algorithm and a local minimisation algorithm. In *Proceedings of the 19th Annual ASME Design Automation Conference, Part 1 (of 2)* (Albuquerque, NM, USA, 1993), vol. 65, pp. 477–484.
- [55] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.
- [56] GENDREAU, M., LAPORTE, G., AND SEMET, F. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & Operations Research* 31 (March 2004), 347–358.
- [57] GILMORE, P. C., AND GOMORY, R. E. A linear programming approach to the cutting-stock problem. *Operations Research* 9, 6 (1961), 849–859.
- [58] GLONEK, G. F. V., AND MCCULLAGH, P. Multivariate logistic models. *Journal of the Royal Statistical Society. Series B (Methodological)* 57, 3 (1995), 533–546.
- [59] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.
- [60] GOLDBERG, D. E., DEB, K., AND KORB, B. Messy genetic algorithms: motivation, analysis and first results. *Complex Systems* 3 (1989), 493–530.
- [61] GOMES, A. M., AND OLIVEIRA, J. F. A GRASP approach to the nesting problem. In *Proceedings of Metaheuristics International Conference (MIC)* (July 2001), pp. 47–52.

- [62] GOMES, A. M., AND OLIVEIRA, J. F. A 2-exchange heuristic for nesting problems. *European Journal of Operational Research* 141 (2002), 359–370.
- [63] GOMES, M. A., AND OLIVEIRA, J. F. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research* 171, 3 (2006), 811–829.
- [64] GÓMEZ, J. C., AND TERASHIMA-MARÍN, H. Approximating multi-objective hyper-heuristics for solving 2D irregular cutting stock problems. In *Proceedings of the 9th Mexican International Conference on Artificial Intelligence Conference on Advances in Soft Computing: Part II* (Berlin, Heidelberg, 2010), MICAI'10, Springer-Verlag, pp. 349–360.
- [65] GOODMAN, E., TETELBAUM, A., AND KUREICHIK, V. A genetic algorithm approach to compaction, bin packing, and nesting problems, 1994.
- [66] HAESSLER, R. W., AND SWEENEY, P. E. Cutting stock problems and solution procedures. *European Journal of Operational Research* 54, 2 (September 1991), 141–150.
- [67] HALAVATI, R., SHOURAKI, S. B., NOROOZIAN, M., AND ZADEH, S. H. Optimizing allocation of two dimensional irregular shapes using an agent based approach. In *Proceedings of World Academy of Science, Engineering and Technology* (2005), vol. 6, pp. 241–244.
- [68] HIFI, M. Exact algorithms for the guillotine strip cutting/packing problem. *Computers & Operations Research* 25, 11 (1998), 925–940.
- [69] HIFI, M., AND M'HALLAH, R. A best-local position procedure-based heuristic for two-dimensional layout problems. *Studia Informatica Universalis, International Journal on Informatics* 2, 1 (2002), 33–56.
- [70] HIFI, M., AND M'HALLAH, R. A hybrid algorithm for the two-dimensional layout problem: the cases of regular and irregular shapes. *International Transactions in Operational Research* 10 (2003), 195–216.
- [71] HOLLAND, J. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.
- [72] HOPPER, E. *Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods*. PhD thesis, School of Engineering, University of Wales, 2000.
- [73] HOPPER, E., AND TURTON, B. C. H. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research* 128 (2000), 34–57.
- [74] HOPPER, E., AND TURTON, B. C. H. A review of the application of meta-heuristic algorithms to 2D strip packing problems. *Artificial Intelligence Review* 16, 4 (2001), 257–300.

- [75] HOPPER, E., AND TURTON, B. C. H. An empirical study of meta-heuristics applied to 2D rectangular bin packing - part II. *Stud. Inform. Univ.* 2, 1 (2002), 93–106.
- [76] HU-YAO, L., AND YUAN-JUN, H. Algorithm for 2D irregular-shaped nesting problem based on the nfp algorithm and lowest-gravity-center principle. *Journal of Zhejiang University SCIENCE A* 7, 4 (2006), 570–576.
- [77] HU-YAO, L., AND YUAN-JUN, H. NFP-based nesting algorithm for irregular shapes. In *Symposium on Applied Computing* (New York, NY, USA, 2006), ACM Press, pp. 963–967.
- [78] HYDE, M. R. *A genetic programming hyper-heuristic approach to automated packing*. PhD thesis, School of Computer Science, Univeristy of Nottingham, 2010.
- [79] JACQUENOT, G., BENNIS, F., MAISONNEUVE, J.-J., AND WENGER, P. 2D multi-objective placement algorithm for free-form components. In *Proceedings of ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference* (December 2009).
- [80] JAIN, A. K., MURTY, M. N., AND FLYNN, P. J. Data clustering: a review. *ACM Computing Surveys* 31, 3 (September 1999), 264–323.
- [81] JAKOBS, S. On genetic algorithms for the packing of polygons. *European Journal of Operational Research* 88, 1 (January 1996), 165–181.
- [82] JALIFF, D., AND DAGNINO, A. An object-oriented tool-kit for building CSP decision support systems. In *IEEE* (1995), pp. 3201–3206.
- [83] JOE, H. Relative entropy measures of multivariate dependence. *Journal of the American Statistical Association* 84, 405 (1989), 157–164.
- [84] JOLLIFFE, I. T. *Principal Component Analysis*, second ed. Springer, October 2002.
- [85] KANDA, J., CARVALHO, A., HRUSCHKA, E., AND SOARES, C. Selection of algorithms to solve traveling salesman problems using meta-learning. *International Journal of Hybrid Intelligent Systems* 8 (August 2011), 117–128.
- [86] KANTOROVICH, L. V. Mathematical methods of organising and planning production. *Management Science* 6 (1960), 366–422.
- [87] KARELAHTI, J. Solving the cutting stock problem in the steel industry. Master’s thesis, Helsinki University of Technology, 2002. Advisor: professor Harri Ehtamo.
- [88] KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. Optimization by simulated annealing. *Science* 220 (1983), 671–680.
- [89] KNUTH, D. E. *Dancing links*, 2000.
- [90] KOHONEN, T., SCHROEDER, M. R., AND HUANG, T. S., Eds. *Self-Organizing Maps*, 3rd ed. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

- [91] KOS, L., AND DUHOVNIK, J. Rod cutting optimization with store utilization. In *International Design Conference* (Dubrovnik, Croatia, May 2000), pp. 313–318.
- [92] LAMOUSIN, H., AND WAGGENSPACK, J. Nesting of two-dimensional irregular parts using a shape reasoning heuristic. *Computer-Aided Design* 29, 3 (March 1997), 221–238.
- [93] LAMOUSIN, H. J., JR, AND DOBSON, G. T. Nesting of complex 2-D parts within irregular boundaries. *Journal of Manufacturing Science and Engineering* 118, 4 (1996), 615–622.
- [94] LEYTON-BROWN, K., NUDELMAN, E., AND SHOHAM, Y. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Principles and Practice of Constraint Programming - CP 2002*, P. Van Hentenryck, Ed., vol. 2470 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2002, pp. 91–100.
- [95] LI, Z., AND MILENKOVIC, V. A compaction algorithm for non-convex polygons and its application. In *SCG '93: Proceedings of the ninth annual symposium on Computational geometry* (New York, NY, USA, 1993), ACM, pp. 153–162.
- [96] LI, Z., AND MILENKOVIC, V. Compaction and separation algorithms for non-convex polygons and their applications. *European Journal of Operations Research* 84 (1995), 539–561.
- [97] LINS, L., LINS, S., AND MORABITO, R. An n-tet graph approach for non-guillotine packings of n-dimensional boxes into an n-container. *European Journal of Operational Research* 141, 2 (September 2002), 421–439.
- [98] LÓPEZ-CAMACHO, E. Hiperheurísticas para resolver el problema de empaqueo irregular de material en dos dimensiones. Master's thesis, Tecnológico de Monterrey, May 2007. Advisor: Dr. Hugo Terashima-Marín.
- [99] LÓPEZ-CAMACHO, E., TERASHIMA-MARÍN, H., AND ROSS, P. Defining a problem-state representation with data mining within a hyper-heuristic model which solves 2D irregular bin packing problems. In *Advances in Artificial Intelligence IBERAMIA* (2010), Á. F. Kuri-Morales and G. R. Simari, Eds., vol. 6433 of *Lecture Notes in Computer Science*, Springer, pp. 204–213.
- [100] LÓPEZ-CAMACHO, E., TERASHIMA-MARÍN, H., AND ROSS, P. A hyper-heuristic for solving one and two-dimensional bin packing problems. In *13th annual conference companion on Genetic and evolutionary computation* (New York, NY, USA, 2011), GECCO '11, ACM, pp. 257–258.
- [101] LÓPEZ-CAMACHO, E., TERASHIMA-MARÍN, H., ROSS, P., AND VALENZUELA-RENDÓN, M. Problem-state representations in a hyper-heuristic approach for the 2D irregular BPP. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation* (New York, NY, USA, 2010), ACM, pp. 297–298.



- [102] LUSS, R., AND D'ASPREMONT, A. Clustering and feature selection using sparse principal component analysis. *CoRR* (2007).
- [103] MAHMOUD, A. F., SAMIA, M., EID, S., AND BAHNASAWI, A. Genetic algorithms for solving 2D cutting stock problem, 2004.
- [104] MARÍN-BLÁZQUEZ, J. G., AND SCHULENBURG, S. Multi-step environment learning classifier systems applied to hyper-heuristics. In *Conference on Genetic and Evolutionary Computation. Lecture Notes in Computer Science* (New York, NY, USA, 2006), ACM, pp. 1521–1528.
- [105] MARÍN-BLÁZQUEZ, J. G., AND SCHULENBURG, S. A hyper-heuristic framework with xcs: learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In *Proceedings of the 2003-2005 international conference on Learning classifier systems* (Berlin, Heidelberg, 2007), IWLCS'03-05, Springer-Verlag, pp. 193–218.
- [106] NAZEMI, J. Kiln Planning, A Cutting Stock Approach. *SSRN eLibrary* (2005).
- [107] NICKLAS, L. D., ATKINS, R. W., SETIA, S. K., AND WANG, P. Y. A parallel solution to the cutting stock problem for a cluster of workstations. In *HPDC '96: Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing* (Washington, DC, USA, 1996), IEEE Computer Society, p. 521.
- [108] NILSSON, C. Heuristics for the traveling salesman problem. Tech. rep., Linköping University, Sweden, 2003.
- [109] OKANO, H. A scanline-based algorithm for the 2D free-form bin packing problem. *Journal of the Operations Research Society of Japan* 45, 2 (June 2002), 145–161.
- [110] ORTIZ-BAYLISS, J. C., ÖZCAN, E., PARKES, A. J., AND TERASHIMA-MARÍN, H. Mapping the performance of heuristics for constraint satisfaction. In *IEEE Congress on Evolutionary Computation'10* (2010), pp. 1–8.
- [111] ORTIZ-BAYLISS, J. C., TERASHIMA-MARÍN, H., ÖZCAN, E., AND PARKES, A. J. On the idea of evolving decision matrix hyper-heuristics for solving constraint satisfaction problems. In *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation* (New York, NY, USA, 2011), ACM, pp. 255–256.
- [112] ÖZCAN, E., BILGIN, B., AND KORKMAZ, E. E. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis* 12, 1 (2008), 3–23.
- [113] PETROVIC, S., AND QU, R. Case-based reasoning as a heuristic selector in hyper-heuristic for course timetabling problems. In *Proceedings of the 6th International Conference on Knowledge-based Intelligent Information Engineering Systems and Applied Technologies (KES)* (2002), vol. 82, pp. 336–340.
- [114] PITSOULIS, L. S., MAURICIO, AND RESENDE, G. C. Greedy randomized adaptive search procedures. In *Handbook of Applied Optimization* (2002), Oxford University Press, pp. 168–183.

- [115] POLI, R., AND GRAFF, M. There is a free lunch for hyper-heuristics, genetic programming and computer scientists. In *EuroGP '09: Proceedings of the 12th European Conference on Genetic Programming* (Berlin, Heidelberg, 2009), L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, and M. Ebner, Eds., vol. 5481 of *LNCS*, Springer-Verlag, pp. 195–207.
- [116] POLI, R., LANGDON, W. B., AND MCPHEE, N. F. *A field guide to genetic programming*. Published via <http://lulu.com> and freely available at <http://www.gp-field-guide.org.uk>, March 2008.
- [117] PONCE-PÉREZ, A., PÉREZ-GARCÍA, A., AND AYALA-RAMÍREZ, V. Bin-packing using genetic algorithms. In *Proceedings of the 15th International Conference on Electronics, Communications and Computers* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 311–314.
- [118] PRANDTSTETTER, M., AND RAIDL, G. R. Combining forces to reconstruct strip shredded text documents, 2008.
- [119] R DEVELOPMENT CORE TEAM. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.
- [120] RAMESH, AND RAMESH. A generic approach for nesting of 2-D parts in 2-D sheets using genetic and heuristic algorithms. *Computer-Aided Design* 33, 12 (October 2001), 879–891.
- [121] RATANAPAN, K., AND DAGLI, C. H. An object-based evolutionary algorithm: The nesting solution, 1998.
- [122] REEVES, C. Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research* 63 (1996), 371–396.
- [123] RICE, J. R. The algorithm selection problem. *Advances in Computers* 15 (1976), 65–118.
- [124] RINGNER, M. What is principal component analysis? *Nature Biotechnology* 26, 3 (March 2008), 303–304.
- [125] RODGERS, J. L., AND NICEWANDER, W. A. Thirteen ways to look at the correlation coefficient. *The American Statistician* 42, 1 (1988), 59–66.
- [126] ROSS, P. Hyper-heuristics. In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, Eds. Springer, New York, 2005, pp. 529–556.
- [127] ROSS, P., AND MARÍN-BLÁZQUEZ, J. G. Constructive hyper-heuristics in class timetabling. *IEEE Congress on Evolutionary Computation* 2 (September 2005), 1493–1500.

- [128] ROSS, P., MARÍN-BLÁZQUEZ, J. G., SCHULENBURG, S., AND HART, E. Learning a procedure that can solve hard bin-packing problems: A new GA-based approach to hyper-heuristics. In *Conference on Genetic and Evolutionary Computation. Lecture Notes in Computer Science* (2003), vol. 2724, Springer-Verlag, pp. 1295–1306.
- [129] ROSS, P., SCHULENBURG, S., MARÍN-BLÁZQUEZ, J. G., AND HART, E. Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In *Conference on Genetic and Evolutionary Computation. Lecture Notes in Computer Science* (San Francisco, CA, USA, 2002), Morgan Kaufmann Publishers Inc., pp. 942–948.
- [130] SCHOLL, A., KLEIN, R., AND JÜRGENS, C. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research* 24 (July 1997), 627–645.
- [131] SMITH, L. I. A tutorial on principal components analysis. Tech. rep., Cornell University, USA, February 26 2002.
- [132] SMITH-MILES, K., AND LOPES, L. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research* 39, 5 (2012), 875–889.
- [133] SMITH-MILES, K. A. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys* 41 (2008), 6:1 – 6:25.
- [134] SMITH-MILES, K. A. Towards insightful algorithm selection for optimisation using meta-learning concepts. In *IEEE World Congress on Computational Intelligence. IEEE International Joint Conference on Neural Networks. IJCNN* (2008), IEEE, pp. 4118–4124.
- [135] SMITH-MILES, K. A., JAMES, R. J., GIFFIN, J. W., AND TU, Y. In *Learning and Intelligent Optimization*, T. Stützle, Ed. Springer-Verlag, Berlin, Heidelberg, 2009, ch. A Knowledge Discovery Approach to Understanding Relationships between Scheduling Problem Structure and Heuristic Performance, pp. 89–103.
- [136] STOYAN, Y., TERNO, J., SCHEITHAUER, G., GIL, N., AND ROMANOVA, T. Phi-functions for primary 2D-objects. *Stud. Inform. Univ.* 2, 1 (2002), 1–32.
- [137] SUGIHARA, K. Measures for performance evaluation of genetic algorithms (extended abstract). In *Proceedings of the 3rd Joint Conference on Information Sciences (JCIS)* (1997), pp. 172–175.
- [138] TERASHIMA-MARÍN, H., FARIÁS-ZÁRATE, C. J., ROSS, P., AND VALENZUELA-RENDÓN, M. A GA-based method to produce generalized hyper-heuristics for the 2D-regular cutting stock problem. In *Conference on Genetic and Evolutionary Computation. Lecture Notes in Computer Science* (New York, NY, USA, 2006), ACM Press, pp. 591–598.
- [139] TERASHIMA-MARÍN, H., FLORES-ÁLVAREZ, E. J., AND ROSS, P. Hyper-heuristics and classifier systems for solving 2D-regular cutting stock problems. In *Conference*

- on Genetic and Evolutionary Computation. Lecture Notes in Computer Science* (New York, NY, USA, 2005), ACM, pp. 637–643.
- [140] TERASHIMA-MARÍN, H., MORÁN-SAAVEDRA, A., AND ROSS, P. Forming hyper-heuristics with GAs when solving 2D-regular cutting stock problems. In *Congress on Evolutionary Computation* (2005), IEEE, pp. 1104–1110.
- [141] TERASHIMA-MARÍN, H., ORTIZ-BAYLISS, J. C., ROSS, P., AND VALENZUELA-RENDÓN, M. Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation* (New York, NY, USA, 2008), ACM, pp. 571–578.
- [142] TERASHIMA-MARÍN, H., AND ROSS, P. Evolution of constraint satisfaction strategies in examination timetabling. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)* (San Mateo, CA, 1999), W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, Eds., Morgan Kaufmann, pp. 635–642.
- [143] TERASHIMA-MARÍN, H., ROSS, P., FARÍAS-ZÁRATE, C. J., LÓPEZ-CAMACHO, E., AND VALENZUELA-RENDÓN, M. Generalized hyper-heuristics for solving 2D regular and irregular packing problems. *Annals of Operations Research* 179 (2010), 369–392.
- [144] TERASHIMA-MARÍN, H., TAVERNIER-DELOYA, J. M., AND VALENZUELA-RENDÓN, M. Scheduling transportation events with grouping genetic algorithms and the heuristic DJD. In *MICAI 2005: Advances in Artificial Intelligence* (2005), A. Gelbukh, I. De Albornoz, and H. Terashima-Marín, Eds., vol. 3789 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 185–194.
- [145] TOUSSAINT, G. T. A simple linear algorithm for intersecting convex polygons. *The Visual Computer* 1 (1985), 118–123.
- [146] TURTON, B., AND HOPPER, E. Application of genetic algorithms to packing problems - a review. In *Proceedings of the Second On-line World Conference of Soft Computing in Engineering Design and Manufacturing* (1997), Springer Verlag, pp. 279–288.
- [147] UDAY, A., GOODMAN, E. D., AND DEBNATH, A. A. Nesting of irregular shapes using feature matching and parallel genetic algorithms. In *Genetic and Evolutionary Computation Conference. Late Breaking Papers* (2001), E. D. Goodman, Ed., pp. 429–434.
- [148] VAZIRANI, V. V. *Approximation Algorithms*. Springer, March 2004.
- [149] VÁZQUEZ-RODRÍGUEZ, J., PETROVIC, S., AND SALHI, A. An investigation of hyper-heuristic search spaces. *CEC 2007. IEEE Congress on Evolutionary Computation* (September 2007), 3776–3783.
- [150] VISWANATHAN, K. V., AND BAGCHI, A. Best-first search methods for constrained two-dimensional cutting stock problems. *Operations Research* 41, 4 (1993), 768–776.

- [151] WANG, P. Y. Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research* 31, 3 (1983), 573–586.
- [152] WANG, W. X. Binary image segmentation of aggregates based on polygonal approximation and classification of concavities. *Pattern Recognition* 31, 10 (1998), 1503–1524.
- [153] WÄSCHER, G., AND GAU, T. Heuristics for the integer one-dimensional cutting stock problem: A computational study. *OR Spectrum* 18 (1996), 131–144.
- [154] WÄSCHER, G., HAUSNER, H., AND SCHUMANN, H. An improved typology of cutting and packing problems. *European Journal of Operational Research. Forthcoming Special Issue on Cutting, Packing and Related Problems* 183, 3 (December 2007), 1109–1130.
- [155] WHELAN, P. F., AND BATCHELOR, B. G. Development of a vision system for the flexible packing of random shapes. In *Machine Vision Applications, Architectures, and Systems Integration, Proc. SPIE* (1992), pp. 223–232.
- [156] WOLPERT, D. H., AND MACREADY, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1 (1997), 67–82.
- [157] XIA, B., AND TAN, Z. Tighter bounds of the first fit algorithm for the bin-packing problem. *Discrete Applied Mathematics* 158, 15 (2010), 1668 – 1675.
- [158] YUE, M. A simple proof of the inequality  $\text{FFD}(L) < 11/9\text{OPT}(L)$ , for all  $L$ , for the FFD bin-packing algorithm. *Acta Mathematicae Applicatae Sinica* 7, 4 (1991), 321–331.
- [159] ZHAO, X., MARRON, J. S., AND WELLS, M. T. The functional data analysis view of longitudinal data. *Statistica Sinica* 14, 3 (2004), 789–808.

