SEP TNM

INSTITUTO TECNOLÓGICO DE TIJUANA

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

CÓMPUTO EVOLUTIVO APLICADO AL NÚCLEO DE LAS HIPERHEURÍSTICAS PARA OPTIMIZAR LAS HEURÍSTICAS GENERADAS EN EL PROBLEMA DE EMPACADO

TRABAJO DE TESIS

Presentado por

M.C. MARCO AURELIO SOTELO FIGUEROA

Para Obtener el Grado de

DOCTOR EN CIENCIAS EN COMPUTACIÓN

Director de Tesis

DR. HÉCTOR JOSÉ PUGA SOBERANES

Tijuana, BC, Febrero, 2015

Declaración de Originalidad

Declaro y prometo que este documento de Tesis es producto de mi trabajo original y no infringe los derechos de terceros, tales como los derechos de publicación, derechos de autor, patentes y similares.

Además, declaro que en las citas textuales que he incluido (las cuales aparecen entre comillas) y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y las publicaciones.

En caso de infracción de los derechos de terceros derivados de este documento de Tesis, acepto la responsabilidad de la infracción y relevo de esta a mi director de tesis, así como al Instituto Tecnologico de Tijuana y sus autoridades.

6 de Febrero 2015, Tijuana, Baja California.

M.C. Marco Aurelio Sotelo Figueroa

To my parents José Reyes Sotelo and Adela Figueroa,
my girlfriend Celic Rivera,
my brothers and
my friends.

Acknowledgements

First I want to thank God for giving me the opportunity to get to this point in my life and in my academic development.

I want to thank to my parents for their unconditional support, they always encouraged me to seek more and always been there when I needed more. To my brothers for those words of encouragement. I would like thank to my girlfriend for her patience, understanding and her support everyday. I am humbled by their support and I always be indebted with them.

I would like to thank Dr. Puga, my thesis supervisor, for making all this possible. Dr. Puga let me explore and experiment with all elements that captured my attention.

In particular Dr. Arturo Hernandez must be thanked for helping me with many techniques that have been used in the present thesis.

I would like thank to the Instituto Tecnologico de León for the opportunity to study the Bachelor, Science Master and now the phD.

I would also like to thank CONACYT for the financial support without which It would not have been able to study this phD and develop this thesis.

Especially thank to my colleagues and phD friends for your friendship and support over these years. Thank you for the Kilometers that we traveled and by all the travels that we enjoy together.

Resumen

La generación de heuristicas para un determinado problema es un proceso que se ha asociado a un Experto y cuando las condiciones del problema cambian es necesario volver a recurrir al experto para poder generar una nueva heuristica para dar una solución al problema.

Algunas veces se pueden utilizar otro tipo de herramientas para tratar de dar una solución al problema, dichas herramientas tratan de suprimir la interacción con el experto. Las Metaheuristicas tratan de dar una solución aproximada mientras que las hyperheuristicas buscan una metodologia para solucionar el problema.

Las HiperHeuristicas usadas para generar heuristicas estaban basadas en Programación Genetica, aunque se ha demostrado que la Programación Genetica tiende a generar soluciones que no en todos sus casos se puede aplicar dado que genera soluciones que incluyen elementos no terminales.

En la presente Tesis se muestra el uso de otras metaheuristicas como motor de busqueda de la Gramatica Evolutiva la cual se utilizo como una HiperHeuristica para la generación de Heurísticas. Los parametros de estas metaheuristicas fueron optimizados mediante un Covering Arrays para poder determinar con cuales valores se obtienen los mejores resultados.

Las heurísticas generadas fueron aplicadas al problema de empacado y los resultados obtenidos por estas heuristicas se compararon contra los obtenidos con las heurísticas clásicas. Los resultados fueron comparados usando la prueba no parametrica de Friedman.

Abstract

The generation of heuristics for a given problem is a process that has been associated to an Expert and when the problem's conditions change is necessary to generate a new heuristic to give a solution to the problem.

Sometimes it is possible to use another kind of tools to try to find a solution, these tools try to suppress the interaction with the Expert. The metaheuristics try to give an approximate solution while hyperheuristicas seek a methodology to solve the problem.

The Hyperheuristic used to generate heuristics were based on Genetic Programming, but it has been shown that the Genetic Programming produces solutions that can't be applied because the solutions includes non-terminal elements.

The present thesis uses other metaheuristics as Search Engine of Grammar Evolution which has been used as HyperHeuristica to generate heuristics. The parameters of these metaheuristics were tuning using Covering Arrays to look up the best values.

The generated heuristics were applied to the bin packing problem and the results were compared against those obtained with classical heuristics. The results were compared using the nonparametric Friedman test.

Contents

List of Figures

List of Tables

List of Algorithms

List of Grammars

Chapter 1

Introduction

The heuristic generation process has been associated by an Expert in a certain area. Sometimes the Expert needs to analyze the problem to determine the main components and propose a ways to solve it, these methods are known as Heuristics. The heuristics is a type of strategy that drastically limits the search for solutions and works in polynomial time, however the heuristics doesn't guarantee optimal solutions; in fact, does not guarantee that you have a solution. Everything that can be said for a useful heuristic is that it offers solutions that are good enough most of the time and depending the instances used, because the heuristic doesn't work equal for all problems instances.

When a problem doesn't have heuristics is necessary to resort to an expert or apply another methodology like metaheuristics. The metaheuristic is a master strategy that guides and modifies other heuristics to produce solutions beyond those typically generated in local search optimization. The metaheuristics are based on physical phenomena, swarm intelligence, evolutionary processes, etc.

If the heuristics and metaheuristics aren't enough to solve a problem, or is necessary to find another kind of solutions, then can be applied the Hyper-Heuristics. An Hyper-Heuristic can be defined as heuristics that search a space of heuristics.

In the present thesis is proposed to use the Hyper-Heuristic to evolve automatically heuristics, using metaheuristics as hyper-heuristic's kernel. The problem used in the present work is the bin packing problem, in this problem have been implemented heuristics and metaheuristics. The Hyper-heuristic's kernels have some parameters that need to be optimized, in this approach the Covery Arrays method has been implemented to discern with which parameter the Hyper-Heuristic can generate the best heuristic. The Grammars used by the Hyper-Heuristic's kernels are proposed, those grammars allows to generated basic heuristics for an instance or for an instance set, also lets to generated heuristics for online or offline instances. The heuristics proposed have different performance over the instances of the art-state, an example of the use of the classical heuristics and the grammatical evolution is shown in Appendix A. The results obtained by the heuristics evolved and the heuristics of the art-state have been compared through the Friedman non-parametric test.

1.1 Motivation

One motivation for the present research is fact that the Genetic Programming has been used as the only way to generate heustics, under the hyper-heuristic concept. Even though exists many representations of the Genetic Programming, like the linear Genetic Programming or Probabilistic Model Building Genetic Programming, all of them use the same optimization paradigm: the Genetic Algorithm.

Although the Bin Packing Problem has been widely studied, there are several researchers trying to find better algorithms or methods that can be apply to this kind of problem. One of this methods applied are the Hyper-Heuristics, either heuristic selection or generation, who are looking to generalize the solutions obtained for a instance set.

1.2 Hypothesis

It's possible to use metaheuristics as hyper-heuristic's kernel to generate heuristics with similar performance, based on the fitness quality, to the best heuristics shown in the art-state.

1.3 Objectives

1.3.1 General Objective

- To generate automatically heuristics that allow for solving several different Bin Packing Problem instances by using Hyper-Heuristics based on metaheuristics.

1.3.2 Specific Objectives

a) To analyze the Hyper-Heuristics based on metaheuristics and the Kernel used in it.

b) To search metaheuristics to be used by Hyper-Heuristics based on metaheuristics.

c) To implement feasible metaheuristics into the Hyper-Heuristic kernell.

d) To generate heuristics for each instances and analyze the results against the results obtained by classical heuristics.

e) To analyze the heuristic components to improve them.

f) To generate heuristics for an instance set, to reduce the heuristics generated for the entire set.

1.4 Methodology

The methodology followed to develop the present investigation was:

First analyze the Bin Packing Problem, the instances used in the art-state, the heuristics and exact algorithms used to try to solve it and the fitness functions used to discern the quality of the solutions. Is necessary to codify the heuristics and obtain the solutions for each instances using differents fitness functions.

Secondly find an alternative to Genetic Programming metaheuristic, analyze the alternatives and the advantages over Genetic Programming. Make test to determinate the best metaheuristic to be used to generate heuristics.

Finally develop an methodology based on hyper-heuristics to generate heuristics for the Bin Packing Problem. Perform test and obtain the heuristics for each instance set and compare the results obtained were used non-parametric test.

1.5 Structure of the Thesis

The thesis is structured as follows: Chapter 2 presents a review of the more relevant literature about the Bin Packing Problem, its instances, heuristics and fitness functions applied. Chapter 3 defines the Hyper-Heuristics and the kind of them. Chapter 4 shows the Genetic Programming and the Grammatical Evolution with several algorithms as search method. Chapter 5 introduces the heuristic generation for each instance by using Grammatical Evolution, the grammar used in this chapter is based on a Genetic Programming work. The Hyper-Heuristic concept is shown in Chapters 6 and 7, in those chapters are generated an heuristic for each instance set, also it was proposed new grammars to improve the results obtained in previous chapters. Chapter 8 includes a clustered version to reduce heuristics generated.

Final remarks, contributions and future work are committed in chapter 9 .

1.6 Academic Publications Produced

The following academic papers and book chapters have been produced as result of the present research. For each publication, is presented one chapter.

1. Chapter 5.

   M. A. Sotelo-Figueroa, H. J. Puga Soberanes, J. Martín Carpio, H. J. Fraire Huacuja, C. L. Reyes, and J. A. Soria-Alcaraz, ”Evolving bin packing heuristic using micro-differential evolution with indirect representation” in Recent Advances on Hybrid Intelligent Systems (O. Castillo, P. Melin, and J. Kacprzyk, eds.), vol. 451 of Studies in Computational Intelligence, pp. 349– 359, Springer Berlin Heidelberg, 2013.

2. Chapter 6.

   M. Sotelo-Figueroa, H. Puga Soberanes, J. Martin Carpio, H. Fraire Huacuja, L. Cruz Reyes, and J. Soria-Alcaraz, ”Evolving and reusing bin packing heuristic through grammatical differential evolution” in Nature and Biologically Inspired Computing (NaBIC), 2013 World Congress on, pp. 92–98, 2013.

3. Chapter 7.

   M. A. Sotelo-Figueroa, H. J. Puga Soberanes, J. M. Carpio, H. J. Fraire Huacuja, L. Cruz Reyes, and J. A. Soria-Alcaraz, ”Improving the bin packing heuristic through grammatical evolution based on swarm intelligence” Mathematical Problems in Engineering, vol. 2014, 2014.

4. Chapter 8.

   M. A. Sotelo-Figueroa, H. J. P. Soberanes, J. M. Carpio, H. J. F. Huacuja, L. C. Reyes, and J. A. S. Alcaraz, ”Clustering bin packing instances for generating a minimal set of heuristics by using grammatical evolution” in Fuzzy Logic

Augmentation of Nature-Inspired Optimization Metaheuristics, pp. 151–162, Springer International Publishing, 2015.

Chapter 2

Bin Packing Problem

2.1 Introduction

The Bin Packing Problem (*BPP*) [1] can be described as follows: given n items that need to be packed in the lowest possible number of bins, each item has a weight w_j, where j is the element; the max capacity of the bins c is also available. The objective is to minimize the bins used to pack all the items, given that each item is assigned only to one bin, and the sum of all the items in the bin can not exceed the bin's size.

This problem has been widely studied, including the following:

a) Proposing new theorems [2, 3].

b) Developing new heuristic algorithms based on Operational Research concepts [4, 5].

c) Characterizing the problem instances [6, 7, 8].

d) Implementing metaheuristics [9, 10, 11, 12].

This problem has been shown to be an NP-Hard optimization problem [13]. A mathematical definition of the BPP is:

Minimize:

$$z = \sum_{i=1}^{n} y_i \tag{2.1}$$

Subject to the following constrains and conditions:

$$\sum_{j=1}^{n} w_j x_{ij} \leq c y_i \qquad i \in N = \{1, \ldots, n\} \tag{2.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j \in N \tag{2.3}$$

$$y_i \in \{0, 1\} \qquad i \in N \tag{2.4}$$

$$x_{ij} \in \{0, 1\} \qquad i \in N, j \in N \tag{2.5}$$

where:

a) w_j: weight of the j item.

b) $y_i = \begin{cases} 1 & \text{if the bin } i \text{ have items} \\ 0 & \text{otherwise} \end{cases}$

c) $x_{ij} = \begin{cases} 1 & \text{if the piece } j \text{ is in the container } i \\ 0 & \text{otherwise} \end{cases}$

d) n: number or available bins.

e) c: capacity of each bin.

The algorithms for the BPP instances can be classified as *online* or *offline* [8]. We have algorithms considered *online* if we don't know the items before starting the packing process, and *offline* if we know all the items before starting. In this research we worked with both algorithms.

2.2 Tests Instances

Beasley[14] proposed a collection of test data sets, known as *OR-Library* and maintained by the Beasley University, which were studied by Falkenauer[15]. This collection contains a variety of test data sets for a variety of Operational Research problems, including the BPP in several dimensions. For the one dimensional BPP case the collection contains eight data sets, that can be classified in two classes:

a) *Unifor* The data sets from binpack1 to binpack4 consist of items of sizes uniformly distributed in $(20, 100)$ to be packed into bins of size 150. The number of bins in the current known solution was found by [15].

b) *Triplets* The data sets from binpack5 to binpack8 consist of items from $(24, 50)$ to be packed into bins of size 100. The number of bins can be obtained dividing the size of the data set by three.

Scholl[16] proposed another collection of data sets, only 1184 problems were solved optimally. Alvim [17] reported the optimal solutions for the remaining 26 problems. The collection contains three data sets:

a) *Set 1* It has 720 instances with items drawn from a uniform distribution on three intervals $[1, 100]$, $[20, 100]$, and $[30, 100]$. The bin capacity is C = 100, 120, and 150 and n = 50, 100, 200, and 500.

b) *Set 2* It has 480 instances with 3, 5, 7 and 9 items. It also includes the following parameters:

- ▷ C = 1000
- ▷ n = 50, 100, 200, 500.
- ▷ $\overline{w}$ = C/3, C/5, C/7, C/9.
- ▷ δ = 20%, 50%, 90%.

This data set includes desired average weight ($\overline{w}$) and the maximal deviation of the single value of $\overline{w}$ (δ). For example, with $\overline{w}$ = C/5 and δ = 20% the weights are randomly chosen from the interval [160, 240].

c) *Set 3* It has 10 instances with C = 100,000, n = 200, and items are drawn from a uniform distribution on [20000, 35000]. Set 3 is considered the most difficult of the three sets.

2.3 Fitness Measure

There are many *Fitness Measures* used to discern the results obtained by heuristics and metaheuristics algorithms, the most simple fitness measure used with the BPP is the number of Bins used (see Eq. (2.6)) . In [18] two Fitness Measures are shown, the first measure (see Eq. (2.7)) tries to find the difference between the used bins and the theorical upper bound on the bins needed. The second (see Eq. (2.8)) was proposed in [10] and rewards full or almost full bins; the objective is to fill each bin, minimizing the free space.

$$Fitness = B \tag{2.6}$$

$$Fitness = B - \frac{\sum_{i=1}^{n} w_i}{C} \tag{2.7}$$

$$Fitness = 1 - \left(\frac{\sum_{i=1}^{n} \left(\frac{\sum_{j=1}^{m} w_j x_{ij}}{C} \right)^2}{n} \right) \tag{2.8}$$

where:

a) B Number of bins used.

b) n Number of containers.

c) m Number of pieces.

d) w_j j-th's piece size.

e) $x_{ij} = \begin{cases} 1 & \text{if the piece } j \text{ is in the container } i \\ 0 & \text{otherwise} \end{cases}$

f) C Bin capacity.

2.4 Exact Methods

Martello and Toth proposed the MTP algorithm [1], this algorithm is based on a First-Fit decreasing. The items are initially sorted by decreasing weights. The algorithm indexes the bins according to the order in which they are initialized.

A decision tree is made, at each node is assigned an item starting with largest, the problem space is bounded by $n!$ where n is the number of elements. The procedures L_2 and L_3 are applied to reduce the current problem, those procedures can be found in [1]. Also is used a backtracking step to removal an item and assign it to the next feasible bin.

This algorithm is have been widely used to obtain the results of instances set, due the BPP is an NP-Hard problem this algorithm can't be applied to large instances.

The results obtained by MTP, using the fitness functions described previously, are shown in Table 2.1.

Instance	Fitness 2.6	Fitness 2.7	Fitness 2.8
bin1data	78378	3727.680200	64.307340
bin2data	20246	280.476930	24.725328
bin3data	562	11.939499	0.390077
binpack1	981	11.866669	0.421557
binpack2	2032	10.513344	0.180042
binpack3	4024	11.293335	0.097509
binpack4	8011	10.873352	0.047260
binpack5	400	0.0	0.0
binpack6	800	0.0	0.0
binpack7	1660	0.0	0.0
binpack8	3340	0.0	0.0
hard28	1972	6.615009	0.167276

Table 2.1: Results obtained from the Test Instances using the Fitness Functions and MTP Algorithm

The MTP algorithm was used with the Fitness Functions shown in Section 2.3 and was applied to solve the Test Instances shown in Section 2.2. The results shown in this table are the total from the results from each instance set.

2.5 Classic Heuristics

Heuristics have been used to solve the BPP, obtaining good results. In [4] shows the following heuristics as Classical Heuristics, these heuristics can be used as *online heuristics* if the items need to be packed as they come in or *offline heuristics* if the items can be sorted before to starting the packing process:

a) *Best Fit* (BF)[19] Puts the piece in the fullest bin that has room for it, and opens a new bin if the piece does not fit in any existing bin.

b) *Worst Fit* (WF)[4] Puts the piece in the emptiest bin that has room for it, and opens a new bin if the piece does not fit in any existing bin.

c) *Almost Worst Fit* (AWF)[4] Puts the piece in the second emptiest bin if that bin has room for it, and opens a new bin if the piece does not fit in any open bin.

d) *Next Fit* (NF)[20] Puts the piece in the right-most bin and opens a new bin if there is not enough room for it.

e) *First Fit* (FF) [20] Puts the piece in the left-most bin that has room for it and opens a new bin if it does not fit in any open bin.

Even though there are some heuristics having better performance than the heuristics shown in the present section [21, 22, 23, 3, 24], such heuristics have been the result of research of Lower and Upper bounds to determine the minimal number of bins.

Instance	AWF	BF	FF	NF	WF
bin1data	21446	20994	20994	23615	21030
bin2data	21446	20994	20994	23615	21030
bin3data	603	596	596	650	596
binpack1	1147	1038	1044	1279	1131
binpack2	2353	2154	2162	2669	2342
binpack3	4626	4240	4255	5300	4614
binpack4	9167	8407	8430	10548	9154
binpack5	420	400	400	400	400
binpack6	820	800	800	800	800
binpack7	1680	1660	1660	1660	1660
binpack8	3360	3340	3340	3340	3340
hard28	2050	1995	1995	2755	2024

Table 2.2: Results obtained using the Fitness Function 2.6 and the online classic heustics

The online classic heuristics were used with the Fitness Functions 2.6 and was applied to solve the Test Instances shown in Section 2.2. The results shown in this table are the total from the results from each instance set.

Instance	AWF	BF	FF	NF	WF
bin1data	1480.476800	1028.477500	1028.477500	3649.477500	1064.477300
bin2data	1480.476800	1028.477500	1028.477500	3649.477500	1064.477300
bin3data	52.939500	45.939500	45.939500	99.939500	45.939500
binpack1	177.866800	68.866800	74.866800	309.866800	161.866800
binpack2	331.513500	132.513300	140.513300	647.513400	320.513500
binpack3	613.293200	227.293200	242.293200	1287.293300	601.293200
binpack4	1166.872900	406.873400	429.873400	2547.873500	1153.872900
binpack5	20.0	0.0	0.0	0.0	0.0
binpack6	20.0	0.0	0.0	0.0	0.0
binpack7	20.0	0.0	0.0	0.0	0.0
binpack8	20.0	0.0	0.0	0.0	0.0
hard28	84.615000	29.615000	29.615000	789.615000	58.615000

Table 2.3: Results obtained using the Fitness Function 2.7 and the online classic heustics

The online classic heuristics were used with the Fitness Functions 2.7 and was applied to solve the Test Instances shown in Section 2.2. The results shown in this table are the total from the results from each instance set.

Instance	AWF	BF	FF	NF	WF
bin1data	67.596600	44.561000	44.562400	110.053900	47.654000
bin2data	67.596600	44.561000	44.562400	110.053900	47.654000
bin3data	1.518900	1.390200	1.390200	2.699700	1.397100
binpack1	5.365600	2.425800	2.604700	7.940900	5.089700
binpack2	4.992200	2.259100	2.397000	7.989200	4.916400
binpack3	4.769700	2.014500	2.133300	7.994800	4.726000
binpack4	4.622200	1.838700	1.935800	7.961100	4.597400
binpack5	1.324500	0.0	0.0	0.0	0.0
binpack6	0.682100	0.0	0.0	0.0	0.0
binpack7	0.335000	0.0	0.0	0.0	0.0
binpack8	0.167100	0.0	0.0	0.0	0.0
hard28	1.927800	0.655500	0.655300	13.133100	1.547100

Table 2.4: Results obtained using the Fitness Function 2.8 and the online classic heustics

The online classic heuristics were used with the Fitness Functions 2.8 and was applied to solve the Test Instances shown in Section 2.2. The results shown in this table are the total from the results from each instance set.

Instance	AWF	BF	FF	NF	WF
bin1data	21446	20994	20994	23615	21030
bin2data	21446	20994	20994	23615	21030
bin3data	603	596	596	650	596
binpack1	1016	995	995	1372	1003
binpack2	2087	2062	2062	2851	2068
binpack3	4100	4078	4078	5647	4085
binpack4	8141	8108	8108	11253	8123
binpack5	479	464	464	491	464
binpack6	936	916	916	971	916
binpack7	1919	1900	1900	2002	1900
binpack8	3823	3801	3801	4024	3801
hard28	2050	1995	1995	2755	2024

Table 2.5: Results obtained using the Fitness Function 2.6 and the offline classic heustics

The offline classic heuristics were used with the Fitness Functions 2.6 and was applied to solve the Test Instances shown in Section 2.2. The results shown in this table are the total from the results from each instance set.

Instance	AWF	BF	FF	NF	WF
bin1data	1480.476800	1028.477500	1028.477500	3649.477500	1064.477300
bin2data	1480.476800	1028.477500	1028.477500	3649.477500	1064.477300
bin3data	52.939500	45.939500	45.939500	99.939500	45.939500
binpack1	46.866800	25.866800	25.866800	402.866800	33.866800
binpack2	65.513300	40.513300	40.513300	829.513400	46.513300
binpack3	87.293200	65.293200	65.293200	1634.293100	72.293200
binpack4	140.873300	107.873300	107.873300	3252.873300	122.873300
binpack5	79.000000	64.000000	64.000000	91.000000	64.000000
binpack6	136.000000	116.000000	116.000000	171.000000	116.000000
binpack7	259.000000	240.000000	240.000000	342.000000	240.000000
binpack8	483.000000	461.000000	461.000000	684.000000	461.000000
hard28	84.615000	29.615000	29.615000	789.615000	58.615000

Table 2.6: Results obtained using the Fitness Function 2.7 and the offline classic heustics

The offline classic heuristics were used with the Fitness Functions 2.7 and was applied to solve the Test Instances shown in Section 2.2. The results shown in this table are the total from the results from each instance set.

Instance	AWF	BF	FF	NF	WF
bin1data	67.596600	44.561000	44.562400	110.053900	47.654000
bin2data	67.596600	44.561000	44.562400	110.053900	47.654000
bin3data	1.518900	1.390200	1.390200	2.699700	1.397100
binpack1	1.479100	0.913900	0.914100	9.504500	1.233400
binpack2	1.043300	0.705900	0.705900	9.459500	0.846500
binpack3	0.746900	0.591600	0.591600	9.412300	0.666500
binpack4	0.622100	0.495700	0.495700	9.404300	0.572300
binpack5	5.453600	4.830800	4.830800	6.420600	4.848600
binpack6	4.993900	4.553700	4.553700	6.197900	4.557000
binpack7	4.775800	4.557000	4.557000	6.078700	4.558700
binpack8	4.527700	4.400500	4.400500	6.059600	4.400700
hard28	1.927800	0.655500	0.655300	13.133100	1.547100

Table 2.7: Results obtained using the Fitness Function 2.8 and the offline classic heustics

The offline classic heuristics were used with the Fitness Functions 2.8 and was applied to solve the Test Instances shown in Section 2.2. The results shown in this table are the total from the results from each instance set.

Chapter 3

Metaheuristics

3.1 Introduction

3.2 Genetic Programming

Genetic Programming (GP) [25, 26, 27, 28] is an evolutionary computation technique that was devised in order to generate programs. It uses a similar concept to Genetic Algorithms (GA) but there are notable differences, particularly what represents each chromosome in genetic algorithms each chromosome represents a possible solution to a problem while on the Genetic Programming each chromosome represents a program or a way to solving a problem.

This metaheuristics became popular in 1992 when John Koza made the publication of his book [26] however there was a history of GP [27] but still did not take the name. The representation by trees was suggested by Cramer [29] and Koza [25] while early work using Lisp and Prolog.

Each node returns its value to its parent node in the usual way the leaf nodes are the input variables that provide information on the status of the problem or are numerical constants, while other nodes or internal nodes running operations leaf nodes (also considered functions).

3.2.1 Genetic Operators

GP have three basic operators which modify the population to generate the next generation, these are: cross, mutation and reproduction, originally in the work of Koza [26] mutation operator is not included yet but with the research has been seen that the mutation may help in the search process and for this reason is included as a standard operator Genetic Programming.

Cross

Standard the operator requires crossing two parents and generates from them two children. A node is chosen at random from each of the parents and are exchanged between parents to generate two sons that are passed to the next generation. Koza suggests choosing an internal node 90% of the time and a leaf node at 10% to make the crosses to generate more complex trees.

Mutation

Mutation operator requires only one parent and produces one child. The node is chosen randomly and a tree is generated at that node. Mutation is a way to diversify the population and help out a local optimum.

Reproduction

The reproduction operator copies a parent exactly one child.

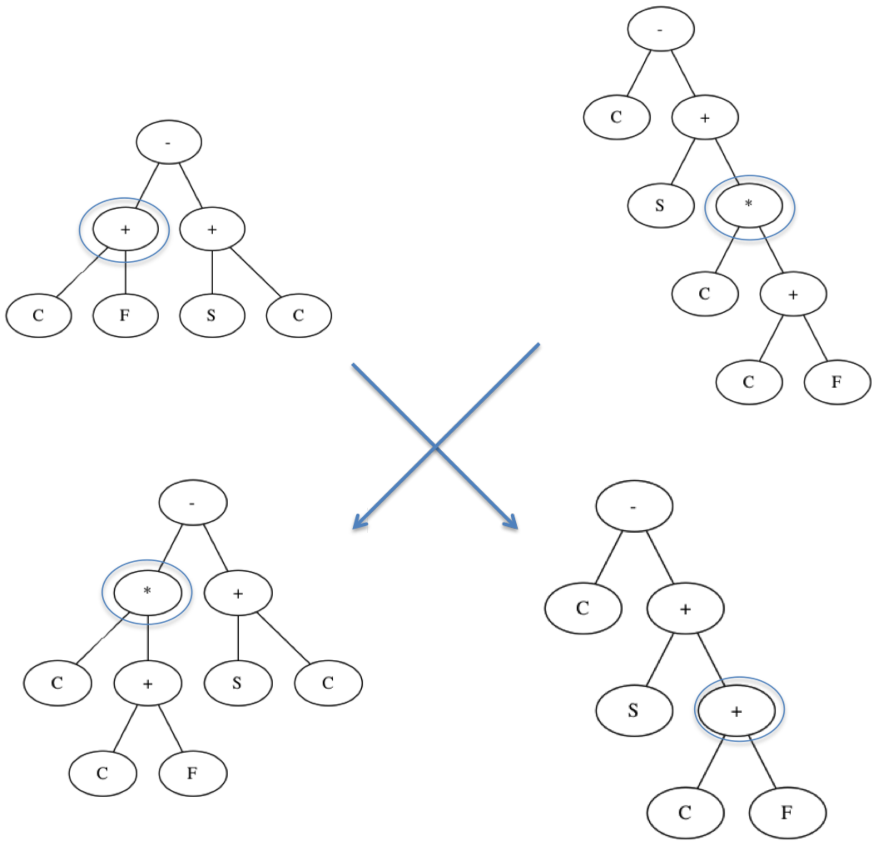

Figure 3.1: GP cross's example

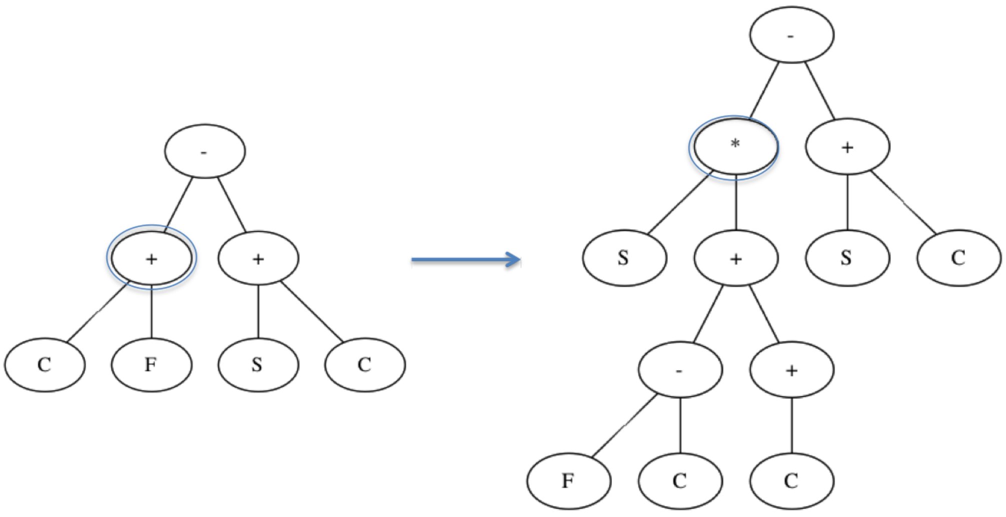

Figure 3.2: GP mutation's example

3.3 Grammatical Evolution

Grammatical Evolution (*GE*) [30] is a grammar-based form of Genetic Programming (*GP*) [31]. GE joins the principles of molecular biology, which are used by GP, and the power of formal grammars. Unlike GP, GE adopts a population of lineal genotypic integer strings, or binary strings, witch are transformed into functional phenotypic through a genotype-to-phenotype mapping process [32], this process is also know as *Indirect Representation* [33]. The genotype strings evolve with no knowledge of their phenotypic equivalent, only using the fitness measure.

The transformation is governed through a Backus Naur Form grammar (*BNF*), which is made up of the tuple N, T, P, S; where N is the set of all non-terminal symbols, T is the set of terminals, P is the set of production rules that map $N \to T$, and S is the initial start symbol where $S \in N$. There are a number of production rules that can be applied to a non-terminal, an "|" (or) symbol separates the options.

Even though the GE uses the Genetic Algorithm (*GA*) [30, 32, 34] as a search strategy it is possible to use another search strategy like the Particle Swarm Optimization, called Grammatical Swarm (*GS*) [35].

In GE each individual is mapped into a program using the BNF, using (3.1) proposed in [32] to choose the next production based-on the non-terminal symbol. An example of the mapping process employed by GE is shown in Figure 3.3.

$$Rule = c\%r \tag{3.1}$$

where c is the codon value and r is the number of production rules available for the current non-terminal.

The GE can use different search strategies; our proposed model is shown in Figure 3.4. This model includes the problem instance and the search strategy as an input. In [32] the search strategy is part of the process, however it can be seen as

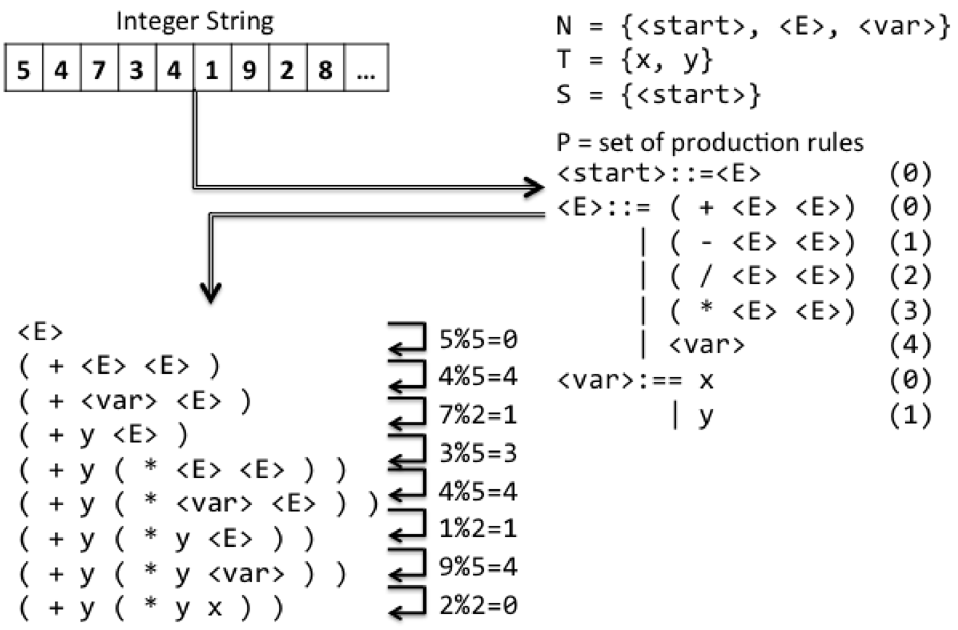

Figure 3.3: GE's mapping process

An example of a transformation from genotype to phenotype using a BNF Grammar. It begins with the start symbol, if the production rule for this symbol is only one rule, then the production rule replaces the start symbol, and the process begins choosing the production rules based on the current genotype. It takes each genotype and the non-terminal symbol from the left to perform the next production using (3.1) until all the genotypes are mapped or there aren't more non-terminals in the phenotype.

an additional element that can be chosen to work with GE. The GE will generate a solution through the search strategy selected and it will be evaluated in the objective function using the problem instance.

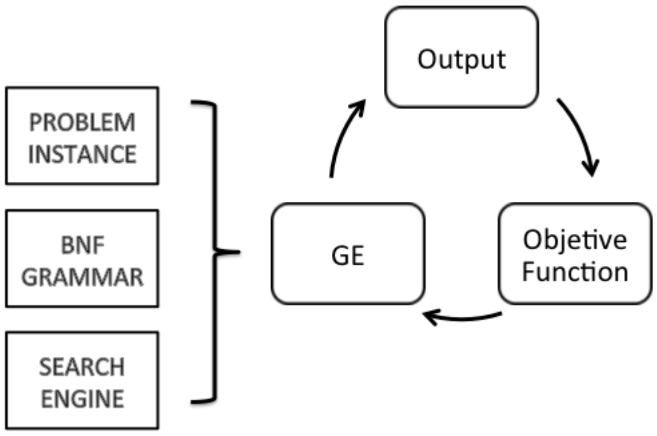

Figure 3.4: GE's methodology

GE's methodology used in the present work, this methodology can be used with different search strategies.

3.3.1 Differential Evolution

Differential Evolution (*DE*) [36] was developed by R. Storn and K. Price in 1996. It is a vector-based evolutionary algorithm, and it can be considered as a further development to Genetic Algorithm (*GA*) [34]. It is a stochastic search algorithm with self-organizing tendency and does not use the information of derivatives[37].

For a d-dimensional problem with d parameters, a population of n solution are initially generated, so we have x_i solution vectors where $i = 1, 2, \ldots, n$. For each solution x_i at any generation t we use the conventional notation as:

$$x_i^t = (x_1^t, x_2^t, \ldots, x_d^t) \tag{3.2}$$

which consist of d-components in the d-dimensional space. This vector can be considered as the chromosomes or genomes.

This metaheuristic consists of three main steps: mutations, crossover and selection.

Mutation is carried out by the mutation scheme. For each vector x_i at any time or generation t, first randomly choose three distinct vector x_p, x_q and x_r at t, and then generate a so-called donor vector by the mutation scheme.

$$v_i^{t+1} = x_p^t + F(x_q^t - x_r^t) \tag{3.3}$$

where $F \in [0, 2]$ is a parameter, often refered to as the scale factor. This requires that the minimum number of population size is $n \geq 4$. We can see that the perturbation $\delta = F(x_q - x_r)$ to the vector x_p is used to generate a donor vector v_i, and such perturbation is directed and self-organized.

The crossover is controlled by a probability $C_r \in [0, 1]$ and actual crossover can be carried out in two ways: binomial and exponential. The binomial scheme performs crossover on each of the d components or variables/parameters. By generating a uniformly distributed random number $r_i \in [0, 1]$, the jth components of v_i is manipulated as:

$$u_{j,i}^{t+1} = \begin{cases} v_{j,i}^t & \text{if } r_i \leq C_r \\ x_{j,i}^t & \text{otherwise} \end{cases} \quad j = 1, 2, ..., d \tag{3.4}$$

This way, each component can be decided randomly whether to exchange with donor vector or not.

In the exponential scheme, a segment of the donor vector is selected and this segment starts with a random k with a random length L which can include many components. Mathematically, this is to choose $k \in [0, d-1]$ and $L \in [1, d]$ randomly, and we have:

$$u_{j,i}^{t+1} = \begin{cases} v_{j,i}^t & v_{j,i}^t \text{ for } j = k, \ldots, k - L \in [1, d] \\ x_{j,i}^t & \text{otherwise} \end{cases} \tag{3.5}$$

The binomial crossover, due to its popularity in many DE literatures [36, 38, 39], is utilized in our implement.

Selection is essentially the same as that used in genetic algorithms. It is to select the fittest, and for minimization problem, the minimum objective value. Therefore, we have:

$$x_i^{t+1} = \begin{cases} u_i^{t+1} & \text{if } f(u_i^{t+1}) \le f(x_i^t) \\ x_i^t & \text{otherwise} \end{cases} \tag{3.6}$$

All the above three components can be seen in the pseudo code as shown in algorithm 3.1. It is worth pointing out there that the use of J_r is to ensure that $v_i^{t+1} \neq x_i^t$, which may increase the evolutionary or exploratory efficiency. The overall search efficiency is controlled by two parameters: the differential weight F and the crossover probability C_r.

Algorithm 3.1 Differential Evolution Algorithm

```
Require: $F$ differential weight, $C_r$ crossover probability, $n$ population size
 1: Initializate the initial population.
 2: while stopping criterion not met do
 3:    for $i = 1$ to $n$ do
 4:       For each $x_i$ randomly choose 3 distinct vector $x_p$, $x_q$ and $x_r$.
 5:       Generate a new vector $v$ by DE scheme (3.3).
 6:       Generate a random index $J_r \in \{1, 2, \dots, d\}$ by permutation.
 7:       Generate a randomly distributed number $r_i \in [0, 1]$
 8:       for $j = 1$ to $n$ do
 9:          For each parameter $v_{j,i}$ (jth component of $v_i$), update
10:          $u_{j,i}^{t+1} = \begin{cases} v_{j,i}^{t+1} & \text{if } r_i \le C_r \text{ or } j = J_r \\ x_{j,i}^t & \text{if } r_i > C_r \text{ or } j \neq J_r \end{cases}$
11:       end for
12:       Select and update the solution by (3.6).
13:    end for
14:    Update the counters such as $t = t + 1$
15: end while
```

In [40] five strategies are shown to mutate the v vector:

a) DE/rand/1: $V_i = X_{r1} + F(X_{r2} - X_{r3})$.

b) DE/best/1: $V_i = X_{best} + F(X_{r1} - X_{r2})$.

c) DE/curren to to best/1: $V_i = X_i + F(X_{best} - X_i) + F(X_{r1} - X_{r2})$.

d) DE/best/2: $V_i = X_{best} + F(X_{r1} - X_{r2}) + F(X_{r3} - X_{r4})$.

e) DE/rand/2: $V_i = X_{r1} + F(X_{r2} - X_{r3}) + F(X_{r4} - X_{r5})$.

where: r_1, r_2, r_3, r_4 are random and mutually different indices, witch should also be different from the trial vector's index i and X_{best} is the individual vector with best fitness.

3.3.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) [41, 42, 43, 44, 45] is a metaheuristic Bio-inspired in the flocks of birds or schools of fish. It was developed by J. Kennedy and R. Eberthart based on a concept called social metaphor. This metaheuristic simulates a society where all individuals contribute with their knowledge to obtain a better solution. There are three factors that influence the change of status or behavior of an individual:

a) The knowledge of the environment or adaptation: it is related to the importance given to the experience of the individual.

b) His Experience or local memory: is related to the importance given to the best result found by the individual.

c) The Experience of their neighbors or Global memory: this is related to how important is the best result obtained by their neighbors or other individuals.

In this metaheuristic each individual is considered as a particle, and moves through a multidimensional space that represents the social space, the search space depends on the dimension of space which in turn depends on the variables used to represent the problem.

For the update of each particle we use the velocity vector which tells how fast the particle will move in each of the dimensions, the method for updating the speed of PSO is given by equation (3.7), and its position is updated by equation (3.8). Algorithm 3.2 shows the complete PSO algorithm.

$$v_i = wv_i + \phi_1 \left(x_i - Bglobal\right) + \phi_2 \left(x_i - Blocal\right) \tag{3.7}$$

$$x_i = x_i + v_i \tag{3.8}$$

where:

a) v_i is the velocity of the i-th particle.

b) w is adjustment factor to the environment.

c) $\phi_1$is the memory coefficient in the neighborhood.

d) ϕ_2 is the coefficient memory.

e) x_i is the position of the i-th particle.

f) B_{global} is the best position found so far by all particles.

g) B_{local} is the best position found by the i-th particle

Algorithm 3.2 Particle Swarm Optimization Algorithm

```
Require: $w$ adaptation to environment coefficient, $\phi_1$ neighborhood memory coefficient, $\phi_2$ memory coefficient, $n$ swarm size.
 1: Start the swarm particles.
 2: Start the velocity vector for each particle in the swarm.
 3: while stopping criterion not met do
 4:   for $i = 1$ to $n$ do
 5:     If the i-particle's fitness is better than the local best then replace the local
        best with the i-particle.
 6:     If the i-particle's fitness is better than the global best then replace the global
        best with the i-particle.
 7:     Update the velocity vector by (3.7).
 8:     Update the particle's position with the velocity vector by (3.8).
 9:   end for
10: end while
```

3.3.3 Particle Evolutionary Swarm Optimization

Particle Evolutionary Swarm Optimization (*PESO*) [46, 47, 48] is based on PSO but introduces two perturbations in order to avoid two problems observed in PSO [49]:

a) Premature convergence.

b) Poor diversity.

Algorithm 3.3 shows the PESO Algorithm with two perturbations, Algorithms 3.4 and 3.5. The C-Perturbation has the advantage of keeping the self-organization potential of the flock as no separate probability distribution needs to be computed meanwhile the M-Perturbation helps keeping diversity into the population.

Algorithm 3.3 Particle Evolutionary Swarm Optimization Algorithm

```
Require: $w$ adaptation to environment coefficient, $\phi_1$ neighborhood memory coefficient, $\phi_2$ memory coefficient, $n$ swarm size.
 1: Start the swarm particles.
 2: Start the velocity vector for each particle in the swarm.
 3: while stopping criterion not met do
 4:     for $i = 1$ to $n$ do
 5:         If the i-particle's fitness is better than the local best then replace the local best with the i-particle.
 6:         If the i-particle's fitness is better than the global best then replace the global best with the i-particle.
 7:         Update the velocity vector by (3.7).
 8:         Update the particle's position with the velocity vector by (3.8).
 9:         Apply the C-Perturbation.
10:         Apply the M-Perturbation.
11:     end for
12: end while
```

Algorithm 3.4 C-Perturbation

```
1: for all Particles do
2:     Generate $r$ uniformly between 0 and 1.
3:     Generate p1, p2 and p3 as random numbers between 1 and the number of particles.
4:     Generate the i-new particle using the following equation and applying it to each particle dimension: $new_i = p1 + r\,(p2 - p3)$.
5: end for
6: for all Particles do
7:     If the i-new particle is better that the i-particle then replace the i-particle with the i-new particle.
8: end for
```

Algorithm 3.5 M-Perturbation

```
1: for all Particles do
2:    for all Dimension do
3:       Generate $r$ uniformly between 0 and 1.
4:       if $r \leq 1/dimension$ then
5:          $new_{id} = random(LowerBound, UpperBound)$
6:       else
7:          $new_{id} = Particle_d$
8:       end if
9:    end for
10: end for
11: for all Particles do
12:    If the i-new particle is better that the i-particle then replace the i-particle
       with the i-new particle.
13: end for
```

3.3.4 Bee Swarm Optimization

Bee Swarm Optimization (BSO) [50, 51] is an hybrid metaheuristic based on Particle Swarm Optimization [41] and the Bee Algorithm (BA) [52]. The BSO uses the PSO and BA elements to try to avoid the problems observed in PSO. The premature convergence is avoided through the radius search and the poor diversity with the scout bees.

The BSO core is the PSO with its equations; speed equation (3.7) and updating equation (3.8). After this is applies the exploration and a search radius from the BA to explore and exploiting. The search radius is based on binary operations, adding and subtracting the radius number to the solution vector as shown in the Figure 3.5. Algorithm 3.6 shows the BSO algorithm used.

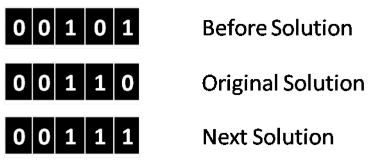

Figure 3.5: Example of search radius based on binary operations.

Algorithm 3.6 Bee Swarm Optimization Algorithm

Require: w adaptation to environment coefficient, ϕ_1 neighborhood memory coefficient, ϕ_2 memory coefficient, n swarm size, sb scout bees, r search radius.

```
 1: Start the bee swarm.
 2: Start the velocity vector for each bee in the swarm.
 3: while stopping criterion not met do
 4:     for $i = 1$ to $n$ do
 5:         If the i-bee's fitness is better than the local best then replace the local best
            with the i-bee.
 6:         If the i-bee's fitness is better than the global best then replace the global
            best with the i-bee.
 7:         Update the velocity vector by (3.7).
 8:         Update the bee's position with the velocity vector by (3.8).
 9:         Apply the search radius $r$ to the local best bee.
10:     end for
11:     Restart the worst $sb$ bees.
12: end while
```

3.3.5 Compact Genetic Algorithm

Compact Genetic Algorithm (cGA) [53, 54, 55] is an improvement over Genetic Algorithm (GA), it was proposed by Harik. Harik analyzed the GA and he treated to model statistically the selection and mutation operators. cGA starts with a probability of 0.5 for each dimension, know as probability vector. cGA isn't a metaheuristic based on population, due the evolution is made adjusting the probability vector.

The cGA can simulate a GA with a given population. The cGA algorithm is shown in Algorithm 3.7.

Algorithm 3.7 Compact Genetic Algorithm

```
Require: $l$ chromosome length, $n$ population size
 1: Initializate the probability vector.
 2: while stopping criterion not met do
 3:    Generate two individuals from the probability vector
 4:    Obtain the Winner and Loser betwen the two individuals
 5:    Update the probability vector
 6:    for $i = 1$ to $l$ do
 7:       if $Winner_i = Loser_i$ then
 8:          if $Winner_i = 1$ then
 9:             $p_i = p_i + \frac{1}{n}$
10:          else
11:             $p_i = p_i - \frac{1}{n}$
12:          end if
13:       end if
14:       Check
15:    end for
16: end while
```

3.3.6 Univariate Marginal Distribution Algorithm

Univariate Marginal Distribution Algorithm (UMDA) [56, 57, 58] is a basic Estimation of Distribution Algorithms (EDA). The UMDA uses a simple model to estimate the joint probability distribution of the selected individuals at each generation, that model is the product of independent univariate marginal distributions as seen in

Equation 3.9. Each univariate marginal distribution is estimated from marginal frequencies using the Equation 3.10. The UMDA's algorithm is shown in Algorithm 3.8.

$$p_l(x) = p_l\left(x|D_{l-1}^{Se}\right) = \prod_{i=1}^{n} p_l(x_i) \tag{3.9}$$

$$p_l(x_i) = \frac{\sum_{j=1}^{n} \delta_j\left(X_i = x_i|D_{l-1}^{Se}\right)}{N} \tag{3.10}$$

where:

a) D_{l-1}^{Se} is the selected population according to the selection method.

b) $\delta_j\left(X_i = x_i|D_{l-1}^{Se}\right) = \begin{cases} 1 & \text{if in the } j^{th} \text{ case of } D_{l-1}^{Se}, X_i = x_i \\ 0 & otherwise \end{cases}$

Algorithm 3.8 Univariate Marginal Distribution Algorithm

```
Require: $l$ chromosome length, $n$ population size, $m$ individuals to be selected.
1: Initializate the initial population randomly.
2: while stopping criterion not met do
3:     Select $m$ individuals according to the selection method.
4:     Estimate the join probability distribution.
5:     Sample the new population.
6: end while
```

Chapter 4

Hyper-Heuristics

4.1 Introduction

Hyper-heuristics (HH) can be defined as heuristics that search a space of heuristics, as opposed to searching a space of solutions directly [59]. The term Hyper-Heuristics is relatively new, however the basic idea has been around since the 1960's. For example, in 1961 (and again in 1963), in [60, 61] is presented an algorithm which combines local job shop scheduling rules using a probabilistic learning technique. This can be classed as a HH because the learning algorithm chooses which of two heuristics to apply, and the chosen heuristic then selects the next job for the machine.

Research in this area is motivated by the goal of raising the level of generality at which optimization systems can operate [62], and by the assertion that in many real-world problem domains, there are users who are interested in good-enough, soon-enough, cheap-enough solutions to their search problems, rather than optimal solutions [62]. In practice, this means researching systems that are capable of operating over a range of different problem instances and sometimes even across problem domains, without expensive manual parameter tuning, and while still maintaining a certain level of solution quality.

The No Free Lunch theorem [63, 64] shows that all search algorithms have the same average performance over all possible discrete functions. This would suggest that it is not possible to develop a general search methodology for all optimization problems as, over all possible discrete functions, no heuristic search algorithm is better than random enumeration [63]. However, it is important to recognise that this theorem is not saying that it is not possible to build search methodologies which are more general than is currently possible. Often the algorithms are developed for a narrower set of problems within that group like for instance university course timetabling [65, 59], exam timetabling problems [66], or bin packing problems [18]. Indeed, algorithms can be specialised further by developing them for a specific problem. At each of these levels, the use of domain knowledge can allow the algorithms to exploit the structure of the set of problems in question. This information can be used to intelligently guide a heuristic search.

In the majority of cases, humans develop heuristics which exploit certain features of a problem domain, and this allows the heuristics to perform better on average than random search. HH research is concerned with building systems which can automatically exploit the structure of a problem they are presented with, and create new heuristics for that problem, or intelligently choose from a set of pre-defined heuristics. In other words, HH research aims to automate the heuristic design process, or automate the decision of which heuristics to employ for a new problem.

The advantage of an automated heuristic design process, is in making optimization tools and decision support available to organisations who currently solve their problems by hand, without the aid of computers. HH research aims to address the needs of organisations interested in good-enough soon-enough cheap-enough solutions to their optimization problems [67]. Note that good enough often means solutions better than they currently obtain by hand, soon enough typically means solutions delivered at least as quick as those obtained by hand, and cheap enough usually means the cost of the system is low enough that its solutions add value to the organisation.

4.2 Heuristic Selection

In the majority of previous work, the HH is provided with a set of human created heuristics. These are often heuristics taken from the art-state, that have been shown to perform well. When using this type of HH approach, the HH is used to choose which heuristic, or sequence of heuristics, to apply, depending on the current problem state.

On a given problem instance, the performance of existing heuristics varies. Therefore, it is difficult to determine which single heuristic will obtain the best result. When using a HH approach, the strengths of the individual heuristics can potentially be automatically combined. However, for such an approach to be worthwhile, the combination should outperform all the constituent heuristics [68]. An optimization system which intelligently chooses heuristics for the problem at hand can be said to operate at a higher level of generality than the individual heuristics. This is because the system can potentially be applied to many different instances of a problem, and maintain its performance.

A possible HH framework is used in [67, 69, 70, 71, 72, 73] . This is shown in Figure 4.1 [74], and has a domain barrier between the domain specific heuristics and the HH. This points out the difference in the responsibilities in the model, between the HH and the domain specific heuristics. The HH maintains only knowledge of how many heuristics it has to call upon, and the results of the evaluation of the solutions they obtain.

This idea enabled a tabu-search HH to be applied to the two very different domains of nurse scheduling and university course timetabling in [75]. Different sets of local search heuristics were used for each of the two problems, but the HH was left unchanged. The HH maintains a ranking of its low level heuristics based on their performance, and applies the one with the highest rank at each decision point. If a heuristic is applied and does not result in a better solution, it is placed in the

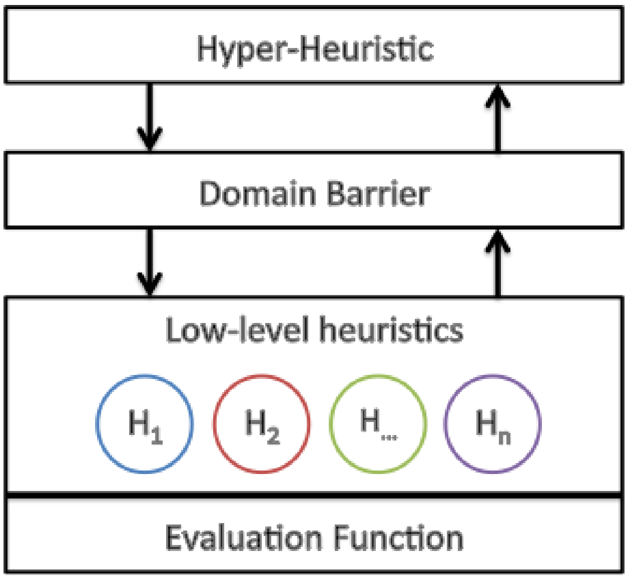

Figure 4.1: Hyper-Heuristic domain barrier

tabu list and therefore is subsequently not used for a number of iterations. The tabu search HH receives no information about the domain in which it is operating. A domain barrier thus makes it possible to apply a HH to a different problem domain without requiring modification. Similar work on a tabu search HH was presented by Kendall and Mohd Hussin [76]. An improved version of this algorithm is outlined in [77], and tested on a real world timetabling problem from the University Technology MARA.

Another example of a HH that maintains a domain barrier is the choice function HH [69, 73]. At each decision point, the choice function evaluates the domain specific heuristics and chooses the best one [69]. The choice function has three terms. The first is a measure of the recent effectiveness of the heuristic, the second is a measure of the recent effectiveness of pairs of heuristics, and the third measures the time since the heuristic was last called. These three terms represent a trade-off between exploration and exploitation, they make it more likely for good heuristics to be used more frequently, but the third term adds the possibility of diversification. Good results are obtained over the domains of sales summit scheduling, presentation scheduling, and nurse scheduling. Applying the choice function on parallel hardware is investigated in [78].

While the domain barrier has been implemented in the framework above, it is not a defining feature of a HH. A HH may or may not be domain specific. Often the HH receives little or no information about the problem domain, but it does receive information about the performance of the heuristics being applied. The heuristics must be domain specific because they must operate on instances of the problem domain to obtain a solution.

Many existing metaheuristics have been employed successfully as HHs. Both a genetic algorithm and a learning classifier system have been applied to the one-dimensional bin packing problem. The learning classifier system HH [79] selects between eight heuristics every time a piece is to be packed. The percentage of pieces left in four different size ranges is calculated, in addition to the percentage of pieces left in relation to the total number of pieces. The problem state is represented as these two features, which are then compared to a set of rules determining which heuristic will be used to pack the next item. This method learns which heuristics should be used when different features are present. Subsequent work evolves similar rule sets using a genetic algorithm HH [68]. This work shows that it is not only a learning classifier system which can generate rules, and that good results can be obtained by assigning reward to rule sets only when the final outcome of the packing is known. Terashima-marín et al. extend this approach onto the two dimensional stock cutting problem by applying a highly similar learning classifier system [80], and genetic algorithm [81]. A comparison of the two HHs for this problem is subsequently given in [82].

Work on a genetic algorithm HH for a trainer scheduling problem is presented in [72, 83, 84, 85]. In this problem, geographically distributed courses are to be scheduled over a period of several weeks, and the courses are to be delivered by a number of geographically distributed trainers [72]. The genetic algorithm chromosome is a sequence of integers that each represent a heuristic. The work is further extended in [72] to incorporate an adaptive length chromosome, so that the length of the sequence of heuristic calls can be modified. Then the results are further improved in [83] by better directing the genetic algorithm in its choice of whether to add or remove genes. A tabu method is added to the genetic algorithm in [84], which means genes are no longer physically added or removed. Instead, they are not used for a number of generations if they do not produce an improvement in the objective function.

An ant algorithm HH is used in [86] for the project presentation scheduling problem. A standard ant algorithm is applied to a search space of heuristics by representing each heuristic as a node in a graph, where an edge between two nodes means one can be applied after the other. The ants traverse the graph, each producing a solution using the heuristic associated with each node they travel through. An ant lays pheromone on the path it took after a full solution has been constructed, in proportion to the quality of the solution. Thus, the good sequences of heuristics become reinforced. Another ant algorithm HH is presented in [87]. The ant colony algorithm optimises a sequence of five heuristics, each with five parameters. Results are obtained on the two dimensional bin packing problem, making them relevant to the work in this thesis. In their earlier work, Cuesta-Canada et al. employ the term HH to refer to the set of five heuristics, rather than to the ant algorithm which searches the space of these heuristics.

Simulated annealing is employed as a HH in [88] for the shipper rationalisation problem, determining space-efficient sizes for reusable containers. The simulated annealing algorithm is based on the tabu search HH presented in [75], with the difference that once the heuristic is selected by the tabu search, the move it generates is accepted according to the simulated annealing algorithm. A simulated annealing HH is also employed in [89] to automate the design of planograms, which are the two dimensional diagrams used to plan shelf space allocation. Greedy and choice function HHs are also investigated in this paper but the simulated annealing had superior performance. Bai et al. present further work inspired by a real world problem in [90], where a collaboration with Tesco informs a study on fresh produce inventory control. Three HHs are implemented for this problem, including the tabu search simulated annealing HH previously presented in [88], and in addition to this, metaheuristic and heuristic approaches are also compared. Issues of memory length in a simulated annealing HH are addressed in [91].

The set of domain specific heuristics that the traditional HH chooses between are usually a combination of mutational and hill climbing heuristics. Including heuristics from these two classes means that the search can explore new areas of the search space, and also exploit good areas. Work by Ozcan et al. in [92] explores three different frameworks, which aim to better make use of the strengths of both classes of heuristic. In the first framework, if the HH chooses a mutational heuristic, a hill climbing heuristic is applied immediately afterwards to exploit the diversification before the next heuristic is chosen. In the second framework, only mutational heuristics are available to the HH, and a single hill climbing heuristic is applied immediately after a mutational heuristic is applied, this framework is similar to a memetic algorithm. The third framework completely separates the sets of mutational and hill climbing heuristics. At each step, the HH chooses and applies a mutational heuristic, and then chooses and applies a hill climbing heuristic.

A HH for timetabling problems is presented in [93], using a tabu search to find a sequence of simple graph colouring heuristics to solve the problem. A genetic algorithm HH is used in [94] for the examination timetabling problem. Using a direct encoding of the problem has been found to be restrictive. For this reason, the genetic algorithm chromosome encodes instructions and parameters for guiding a search algorithm, rather than encoding a particular solution. An example of the limitations of a particular direct representation can be found in [95].

Bilgin et al. investigate combinations of seven heuristic selection mechanisms and five move acceptance criteria [96], and the results are also tested on examination timetabling benchmarks. This work shows that many HHs in the literature can be viewed as a combination of one heuristic selection mechanism and one acceptance criteria, and that a different combination of these components can be classed as a different HH. Ersoy et al. present similar work using a HH to choose between hill climbers in a memetic algorithm [97]. This, again, shows that a HH can be used to optimise a component of an existing metaheuristic algorithm, not just to choose between fixed algorithms.

Two genetic algorithm HH strategies are analysed for the job shop scheduling problem in [98]. One strategy evolves the choice of which priority rule from twelve to use for conflict resolution at a given iteration. The other evolves the sequence in which the machines are considered by a shifting bottleneck heuristic. This can be called a HH because the space of ordering heuristics is searched for one which minimises the makespan. Storer et al. present work which is highly relevant to hyper-heuristic research, also on the job shop scheduling problem. They state, Search spaces can also be generated by defining a space of heuristics [99], which is a fundamental principle of HH research. Specifically, they do this by parameterising existing heuristics and searching the space of parameters. Their subsequent research on how to successfully search such a space is presented in [100].

Hart and Ross have also developed a HH approach for the job shop scheduling problem [101]. They use a genetic algorithm, where each gene represents a combination of scheduling algorithm and heuristic. The scheduling algorithm generates a set of schedulable operations, and the heuristic determines which among those will be chosen. Zhang and Dietterich use reinforcement learning to learn heuristics for the job shop problem, and their approach is tested on a NASA space shuttle payload processing problem, as well as an artificial problem set. Fang et al. present a genetic algorithm for the job shop scheduling problem, which can be labelled a HH as the genome represents an ordering of the jobs [33]. A subsequent proposed extension to the open shop scheduling problem, in the same paper, involves the genome representing an ordering of both the jobs and the specific tasks that they consist of. They later present the results of this extension in [102]. This paper further expands the HH theme of the work, by encoding into the genome the choice of heuristic that is used to schedule each specific task. In the previous work, a fixed heuristic was used and only the ordering was evolved. The chromosome encodes pairs of values representing which heuristic from eight to apply to which remaining job. The heuristic chooses an operation from the job and places it at the earliest time available in the solution.

Cowling and Chakhlevitch [103] state that the performance of a HH relies very much on the choice of low level heuristics. They address the question of how the set of low level heuristics should be designed. Related to this, is the problem of ensuring that the set of low level heuristics is varied enough to ensure an efficient search, but not so large that it contains heuristics that are not necessary. Further work by the same authors addresses this problem, by introducing learning mechanisms into a HH to avoid using heuristics which do not make valuable contributions [104]. This reduces computational effort, because the under performing heuristics do not then slow the search down.

A case based reasoning HH is presented in [67] for the course timetabling problem, and then again in [105], where it is shown that the HH can operate over both the exam timetabling and the course timetabling domains. This HH works by comparing the current problem to problems encountered before, and applying the same heuristics that have worked well in similar situations. Tabu search is employed in [106] to search for the best combination of two well known graph based heuristics for constructing exam timetabling solutions. The tabu search mechanism is based on that presented in [75]. The knowledge gained from this HH is then used to inform two hybrid graph based approaches to the same problem, one which inserts a certain percentage of one heuris- tic into the heuristic list, and one in which case based reasoning remembers heuristics that have been successfully used on similar partial solutions.

Highly related to the case based reasoning approach is the COMPOSER algorithm [107]. COMPOSER essentially consists of a set of condition-action rules which are learned while working through a representative training set [107]. This algorithm is applied to deep space network scheduling in [108, 109]. The algorithm developed by Fink [110] chooses among three solution methods which apply heuristics in different ways, and is inspired by the PRODIGY architecture that COMPOSER utilises. The solution methods are chosen based on a statistical analysis of their past performance. This is different to case based reasoning, as it does not rely on any prior knowledge of a particular problem domain. This approach focuses on selecting a method to be used throughout the whole search, with no opportunity for switching methods while the search is performed.

Pisinger and Ropke present a HH that can operate over five different variants of the vehicle routing problem [111]. The approach employs adaptive large neighbourhood search, a method which selects a heuristic to destroy part of the solution and a heuristic to rebuild it. Adaptive large neighbourhood search is a paradigm rather than a specific algorithm, it requires an acceptance criteria to be specified as well as the heuristics which drastically modify the solution. The acceptance criteria used in this paper is simulated annealing, but any metaheuristic could be used. The paradigm has similarities to variable neighbourhood search [112], which can also be considered a HH. This is because adaptively changing the neighbourhood of the search can be seen as intelligently selecting an appropriate heuristic.

Wah presents a population based method of learning by examples [113], to create new heuristics. This work led to the development of the TEACHER (TEchniques for the Automated Creation of HEuRistics) learning system, which uses similar genetics based learning to design heuristics [114, 115, 116]. The system is for use in knowledge lean environments, and as such is highly related to the work presented here. An important part of the system is the partitioning of the problem domain into subclasses before creating a heuristic for each. This relates directly to results in this thesis, which show that it is beneficial to evolve a heuristic on a subclass of problems, representative of those on which it will be used in the future. In [114], the system is applied to improve two circuit testing solvers, and more examples of TEACHER applications are given in [117, 114, 118].

The concept of a heuristic to choose heuristics is closely related to the concept of multimeme algorithms [119]. In a memetic algorithm, a meme represents a local search heuristic or some other move operator. The results in [119] show the benefits of using more than one meme when performing a search. This can also be thought of as a HH employing a number of low level heuristics during a search.

4.3 Heuristic Generation

Less well studied in the literature is the class of HHs which aim to create a heuristic from a set of potential components. This is distinct from the previous section, in which the HH is given a set of complete functioning heuristics, and must then choose between them. The created heuristic may be ‘disposable’ in the sense that it is created just for one problem instance, and is not intended for use on unseen instances. Alternatively the heuristic may be created for the purpose of reusing it on new unseen instances of a certain problem class.

There are a number of potential advantages of this approach. Every problem instance is different, and obtaining the best possible result for an instance would ideally require a new heuristic or a variation of a previously created heuristic. It would be inefficient for a human analyst to create a new heuristic for every problem instance. Human created heuristics are rarely applicable to only one problem instance, they are usually created for a class of instances, or for all problem instances.

For example, best-fit is a human created heuristic for one dimensional bin packing

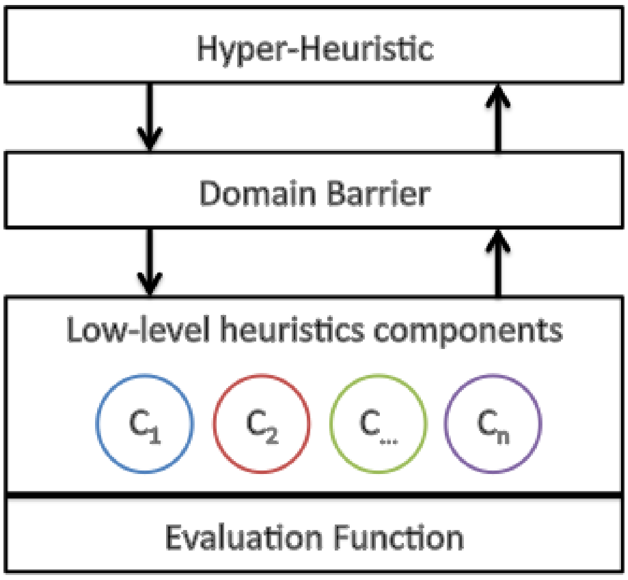

Figure 4.2: Hyper-Heuristic domain barrier used to generate heuristics

[120], and performs well on a wide range of bin packing instances. Indeed, it was created as a general heuristic for any bin packing instance. Over a narrower set of instances however, with piece sizes defined over a certain distribution, best-fit can be outperformed by heuristics which are tailored to the distribution of piece sizes [121]. Poli, Woodward and Burke [122] also employ genetic programming to evolve heuristics for bin packing. The structure within which their heuristics operate is based on matching the piece size histogram to the bin gap histogram, and is motivated by the observation that space is wasted when placing a piece in a bin leaves a smaller available space than the size of the smallest piece still to be packed. Their work also differs from that presented in this thesis because they use linear genetic programming, and the problem addressed is offline bin packing rather than online. However, the motivation for the work is the same, to automate the process of designing heuristics for bin packing.

If the heuristic design process is automated, a computer system could produce a good quality heuristic for an instance in a practical amount of time. This heuristic could even produce a solution that may be better than that which can be obtained by any current human created heuristic. This is because it would have been created specifically for that instance rather than as a general heuristic. Automating the heuristic design process offers the chance to easily, and flexibly, specify the range of instances that a heuristic will be applicable to, and then obtain that heuristic with minimal human effort spent in the process.

A good heuristic for a given problem instance may be counterintuitive, especially if the instance is complex. Another advantage of automated heuristic design, therefore, is that a heuristic may be created which performs well on the instance, but was unlikely to have been created by a human analyst.

Fukunaga presents an automated heuristic discovery system, for the SAT problem, in [123, 124]. The most successful heuristics for SAT can be expressed as different combinations of a particular set of components. It is stated that human researchers are particularly good at identifying good components, but the process of combining them could benefit from automation [123]. The paper uses an evolutionary algorithm to evolve human competitive heuristics consisting of these components. Some issues and potential objections from [123]are resolved in [124],by implementing the system in Lisp for quicker evaluation, and showing that good heuristics can still be evolved without initialising the population with complex hand coded heuristics and components.

Bader-El-Din and Poli [125] observe that the approach of Fukunaga results in heuristics consisting of other nested heuristics. This results in heuristics which are composites of those in early generations, and which are therefore relatively slow to execute. Bader-El-Din and Poli present a different heuristic generation methodology for SAT, which makes use of traditional crossover and mutation operators to produce heuristics which are more parsimo- nious, and faster to execute. A grammar is defined, which can express four existing human created heuristics, and allows significant flexibility to create completely new heuristics. Subsequent research by the same team of researchers has shown that a HH system can evolve constructive heuristics for timetabling problems [126]. The evolved heuristics are not intended to be reusable, and the system is presented as an online learning method, which can often obtain better results than other constructive algorithms and HH approaches.

Keller and Poli present a linear genetic programming HH for travelling salesman problems in [127]. At its simplest level, the system evolves sequences of 2-opt and 3-opt swap heuristics. Then conditional and loop components are added to the set of components, to add complexity to the evolved heuristics. Work presented by Ho and Tay in [128, 129], employs genetic programming to evolve composite dispatching rules for the job shop scheduling problem. The terminals of the genetic programming algorithm are components of previously published dispatching rules, and the functions are the arithmetic operators and an automatically defined function (ADF, see [82]). The evolved dispatching rules are functions, which assign a score to a job based on the state of the problem. When a machine becomes idle, the dispatching rule is evaluated once for each job in the machine's queue, and each result is assigned to the job as its score. The job in the queue with the highest score is the next job to be assigned to the machine. Jakobovic et al. employ the same technique for the parallel machine scheduling problem [130].

Dimopoulos and Zalzala [131] evolve priority dispatching rules for the single machine scheduling problem, to minimise the total tardiness of jobs. The terminal set is based on the human designed 'Montagne' dispatching rule, and contains five elements, representing both global and local job information. The function set consists of the four basic arithmetic operators. While the function and terminal sets are relatively simple, the system evolves heuristics superior to the Montagne, ADD, and SPT heuristics. Genetic programming is also used to evolve priority dispatching rules in [132]. ADFs are not used in this paper, however the function and terminal sets are expanded from that presented in [131, 128]. Human competitive heuristics are produced, even for job shop environments with multiple machines, where a unique dispatching rule is evolved for each.

Recent work by Kumar et al. presents a genetic programming system which evolves heuristics for the biobjective knapsack problem [133]. This is the first work in which heuristics for a multiobjective problem have been automatically generated with a HH. The technique employs many similar ideas to the bin packing methodology presented in this thesis. For example, the terminals used are similar, and the framework which iterates through the pieces, applying the heuristic to each. Further work has shown that heuristics can also be generated with genetic programming for a multiobjective minimum spanning tree problem [134].

Oltean [135] presents a linear genetic programming HH, which generates evolutionary algorithms. A standard individual in a linear genetic program represents a series of instructions that manipulate values in a memory array. The memory positions are referred to as registers. Oltean represents an evolutionary algorithm population as this memory array, with each member of the population stored in one register. The genetic operators are the instructions that operate on the memory array. Tavares et al. [136] also present a methodology for evolving an evolutionary algorithm. They specify the main components of a generic evolutionary algorithm, including initialisation, selection, and the use of genetic operators. Tavares et al. explain how each of these steps, can be evolved individually by a genetic programming HH. They demonstrate the approach through an example of evolving an effective mapping function of genotype to phenotype, for a function optimization problem.

Pappa and Freitas employ a grammar based genetic programming system to evolve rule induction algorithms for classification [137]. They show that classification algorithms can be automatically specialised to one of two problem classes, a conclusion which is similar to some of the conclusions of this thesis. Their paper states that automatic generation of algorithms may result in better algorithms than have currently been designed by hand, which is a common motivation for HH research. They also assert that such an approach can be cheaper than manual algorithm design. As explained previously in this section, this reduction in cost is another reason why HH research is potentially beneficial to smaller organisations.

The literature also shows a link between memetic algorithms and HHs. Krasnogor states a definition of a true memetic algorithm as “evolutionary algorithms where not only the solutions to the problem at hand are evolved but where local searchers that can further improve the quality of those solutions are coevolved, e.g., by GP, alongside them” [119]. Further work by Krasnogor and Gustafson has shown that this is possible, with self generation of memes. For protein structure prediction, a grammar is defined in [138, 139], which expresses groups of memes, each of which performs a local search at a given point in the genetic algorithm. All aspects of the memes are evolved, and the result is a memetic algorithm which is evolved with genetic programming. This provides the possibility of automatically tuning the metaheuristic (in this case a memetic algorithm) to any problem instance [139], and replacing time consuming manual algorithm tuning.

Chapter 5

Evolving Heuristics

5.1 Introduction

This chapter presents the initial work to develop heuristics automatically for the bin packing problem, is based on [140] where the GP is used to evolve heuristics.

The aims of the present chapter was to investigate if is possible to generate heuristics using something different to GP and was necessary to obtain at least the same results obtained by the Genetic Programming. The GE was used and it was proposed a Grammar based on the elements used by the GP.

The results obtained with the GE, using differents kernels, was compared against the results obtained by the classical heuristics, shown in Section 2.5 ,by using Friedman non-parametric test.

5.2 Methodology

The proposed approach tries to find an heuristic, which is developed through the evolution of the grammar by GE with Search Engines. It was used the methodology shown in Figure 5.1, it was generated one heuristic by each instance and the heuristic components were taken from Table 5.1 to make the Grammar 5.1.

The Table 5.2 contains the parameters used for each Search Engine. Those parameters were obtained through a fine tuning parameter using Covery Arrays.

33 experiments were performed independently and the median was used to compare the results against those obtained with the heuristics described in Section 2.5. The comparison was implemented through the non-parametric test of Friedman [141, 142] , this non-parametric test used a post-hoc analysis to discern the performance between the experiments and gives a ranking of them.

The Appendix A includes an Example to pack items based on classic heuristics and GE.

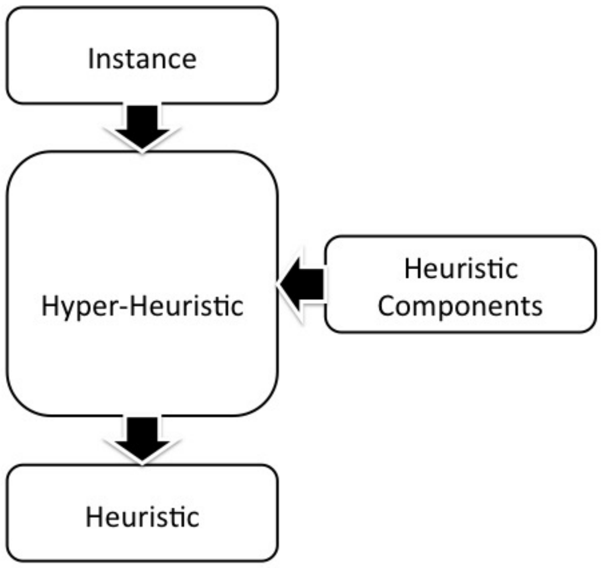

Figure 5.1: Methodology used to Evolve Heuristics

$$
\begin{array}{rcl}
\langle\text{inicio}\rangle & \models & (\langle\text{expr}\rangle) <= (\langle\text{expr}\rangle) \\
\langle\text{expr}\rangle & \models & (\langle\text{expr}\rangle\langle\text{op}\rangle\langle\text{expr}\rangle) \mid \langle\text{var}\rangle \mid \mathit{abs}(\langle\text{expr2}\rangle) \\
\langle\text{expr2}\rangle & \models & (\langle\text{expr2}\rangle\langle\text{op}\rangle\langle\text{expr2}\rangle) \mid \langle\text{var}\rangle \\
\langle\text{var}\rangle & \models & \text{F} \mid \text{C} \mid \text{S} \\
\langle\text{op}\rangle & \models & + \mid * \mid - \mid /
\end{array}
$$

Grammar 5.1: Grammar based on the Heuristic Components as shown in Table 5.1

	Symbol	Arguments	Description
Functions	+	2	Add
	-	2	Subtract
	*	2	Multiply
	/	2	Divide
	$\leq$	2	Tests if the first argument is less than or equal to the second argument.
Terminals	F	0	Returns the sum of the pieces already in the bin
	C	0	Returns the bin capacity
	S	0	Returns the size of the current piece

Table 5.1: Heuristic Componets usign by GP

Parameter	DE	PESO	PSO	UMDA
Functions Call	1500			
Population Size	50			
Strategie	DE/rand/1	-	-	-
F	0.9	-	-	-
C_r	0.8	-	-	-
w	-	1	1	-
ϕ_1	-	0.8	0.8	-
ϕ_2	-	0.5	0.5	-

Table 5.2: Parameters obtained by Covery Arrays to be used by GE

5.3 Results

The Table 5.3 shows an example of the heuristics obtained, there are shown only one example per instance set even though were obtained one heuristic by instance. The generated heuristics were applied as follows: once the heuristic was obtained by the GE and it was analyzed if it only contains terminal components, each item from the instances is proved with the heuristic to decide if the item will be packed into the bin.

The Table 5.4 shows the results obtained with GE, it shows the results by each Search Engine. The results were clustered to show the results by instances set. It was performed the Friedman non-parametric test to discern the results obtained, the Table 5.5 shows the ranking compute by the post-hoc procedure. The value of the non-parametric test is 63.03125 and the p-value 3.42977E-8.

Instance	Heuristic Generated
bin1data	$((S - (F - (F + F)))) \leq (abs(C))$
bin2data	$((F + S)) \leq (C)$
bin3data	$((abs(F) + S)) \leq (abs(C))$
binpack1	$(F) \leq (abs((C - S)))$
binpack2	$(abs((F + S))) \leq (abs((S - (S - C))))$
binpack3	$((abs(S) + F)) \leq (C)$
binpack4	$(abs((S + (S - S)))) \leq (abs((F - C)))$
binpack5	$(S) \leq ((C - F))$
binpack6	$(F) \leq (abs((S - C)))$
binpack7	$(abs(F)) \leq (abs((((C / F) * F) - S)))$
binpack8	$(S) \leq (abs((C - (C / (C / F)))))$
hard28	$((F + S)) \leq (C)$

Table 5.3: Example of the heuristic generate with the Grammar 5.1

Instance	DE	PESO	PSO	UMDA
bin1data	316.106050	316.106050	316.106050	316.106050
bin2data	93.007030	93.007030	93.025230	93.007030
bin3data	1.885825	1.885825	1.885825	1.885825
binpack1	2.425894	2.425894	2.425894	2.425894
binpack2	2.259154	2.259154	2.259154	2.259154
binpack3	2.014334	2.014334	2.014334	2.014334
binpack4	1.838669	1.838669	1.838669	1.838669
binpack5	0.000000	0.000000	0.000000	0.000000
binpack6	0.000000	0.000000	0.000000	0.000000
binpack7	0.000000	0.000000	0.000000	0.000000
binpack8	0.000000	0.000000	0.000000	0.000000
hard28	0.655480	0.655480	0.655480	0.655480

Table 5.4: Results obtained with evolving heuristics by using GE with different Search Engine

Algorithm	Friedman
MTP	2.833333
BF	5.583333
Offline BF	5.791667
Offline FF	5.875000
FF	6.416667
DE	7.875000
PESO	7.875000
UMDA	7.875000
PSO	8.041667
Offline WF	8.083333
WF	8.083333
Offline AWF	9.500000
AWF	10.833333
NF	10.833333
Offline NF	14.500000

Table 5.5: Algorithm ranking based on a Friedman non-parametric test post-hoc procedure to discern the results obtained by Evolving Heuristics

5.4 Results Analysis

It has been shown that human designed heuristics like FF can be easily obtained by GE. The heuristics were evolved without any apriori knowledge. It was only used a grammar to represent the problem.

The proposal metaheuristics used as Search Engine gives the same performances, except the PSO due it doesn't include exploration and exploitation. The UMDA doesn't need to fine tuning its parameters and it allows to obtain same results than the PESO and DE with parameter tuning.

Although is generated one heuristic by instance this methodology can be considerate as HH due the generality, in all cases is used the same Grammar and the methodology is applied in the same way for all instances.

For all cases were obtained heuristics that can be applied to a BPP instance. The objective function used is capable to discern in a better way since it measures the available space of the bins instead the number of bins.

Chapter 6

Evolving and Reusing Heuristics

6.1 Introduction

The previous Chapter shown that is possible to generate automatically heuristics for the BPP, however those heuristics were made for each instances.

In the present Chapter is shown a methodology to generate heuristics, this methodology is based on the HH concept to generate one heuristic by instance set. The aim is to generate an heuristic that can solve in average a instance set.

The results obtained with the GE, using differents kernels, was compared against the results obtained by the classical heuristics, shown in Section 2.5 ,by using Friedman non-parametric test.

6.2 Methodology

The Chapter 5 shows that is possible to generate a heuristic by an instance with performance like the FF Heuristic. The new approach tries to rise the level of generality of the methodology, it generates one heuristic by instance set, as can see in Figure 6.1. It was proposed the Grammar 6.1, this Grammar allows to sort the bins like the BF heuristic.

The parameters used for each Search Engine are shown in Table 5.2, the parameters obtained for the previous test doesn't change with the new Grammar.

33 experiments were performed independently and the median was used to compare the results against those obtained with the heuristics described in Section 2.5. The comparison was implemented through the non-parametric test of Friedman [141, 142] , this non-parametric test used a post-hoc analysis to discern the performance between the experiments and gives a ranking of them.

The Appendix A includes an Example to pack items based on classic heuristics and GE.

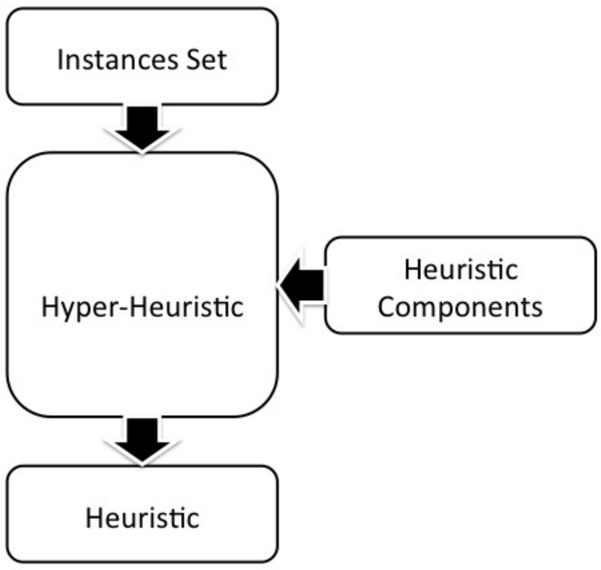

Figure 6.1: Methodology used to Evolve Heuristics based on HH concept

$$\begin{array}{rcl}
\langle\text{inicio}\rangle & \models & \langle\text{exprs}\rangle.(\langle\text{expr}\rangle) <= (\langle\text{expr}\rangle) \\
\langle\text{exprs}\rangle & \models & Sort(\langle\text{exprk}\rangle, \langle\text{order}\rangle) \ | \ \lambda \\
\langle\text{exprk}\rangle & \models & \text{Bin} \ | \ \text{Content} \\
\langle\text{order}\rangle & \models & \text{Asc} \ | \ \text{Des} \\
\langle\text{expr}\rangle & \models & (\langle\text{expr}\rangle\langle\text{op}\rangle\langle\text{expr}\rangle) \ | \ \langle\text{var}\rangle \ | \ abs(\langle\text{expr2}\rangle) \\
\langle\text{expr2}\rangle & \models & (\langle\text{expr2}\rangle\langle\text{op}\rangle\langle\text{expr2}\rangle) \ | \ \langle\text{var}\rangle \\
\langle\text{var}\rangle & \models & \text{F} \ | \ \text{C} \ | \ \text{S} \\
\langle\text{op}\rangle & \models & + \ | \ * \ | \ - \ | \ /
\end{array}$$

Grammar 6.1: Grammar proposal to obtain results like the obtained by the BF Heuristic.

6.3 Results

The Table 6.1 shows an example of the heuristics obtained, there are shown only one example per instance set.

The Table 6.2 shows the results obtained with GE, it shows the results by each Search Engine. It was performed the Friedman non-parametric test to discern the results obtained, the Table 6.3 shows the ranking compute by the post-hoc procedure. The value of the non-parametric test is 76.77708 and the p-value 1.73075E-10.

Instance	Heuristic Generated
bin1data	Sort (Content , Asc) . (F) $\leq$ ((C - S))
bin2data	Sort (Content , Des) . (F) $\leq$ ((C - abs(S)))
bin3data	Sort (Content , Asc) . ((F + S)) $\leq$ (C)
binpack1	Sort(Content, Des) . (abs(F)) $\leq$ ((C - abs(S)))
binpack2	Sort(Content, Des) . ((F + S)) $\leq$ (C)
binpack3	Sort(Content, Des) . (F) $\leq$ (abs((C - S)))
binpack4	Sort(Content, Asc) . (S) $\leq$ ((C - F))
binpack5	Sort(Content, Des) . ((S + F)) $\leq$ (C)
binpack6	Sort(Content, Des) . (F) $\leq$ ((abs(C) - S))
binpack7	Sort(Content, Des) . (abs(F)) $\leq$ (abs((S - C)))
binpack8	Sort(Content, Des) . (abs((S + F))) $\leq$ (C)
hard28	Sort (Content , Asc) . (abs((S + F))) $\leq$ (abs(C))

Table 6.1: Example of the heuristic generate with the Grammar 6.1

Instance	DE	UMDA	PESO	PSO
bin1data	47.545100	47.545100	47.545100	47.545100
bin2data	44.561000	44.561000	44.561000	44.561000
bin3data	1.390200	1.390200	1.390200	1.390200
binpack1	2.425800	2.425800	2.425800	2.604700
binpack2	2.259100	2.259100	2.259100	2.259100
binpack3	2.014500	2.014500	2.014500	2.014500
binpack4	1.838700	1.838700	1.838700	1.838700
binpack5	0.000000	0.000000	0.000000	0.000000
binpack6	0.000000	0.000000	0.000000	0.000000
binpack7	0.000000	0.000000	0.000000	0.000000
binpack8	0.000000	0.000000	0.000000	0.000000
hard28	0.655500	0.655500	0.655500	0.655500

Table 6.2: Results obtained with evolving heuristics and reusing it by using GE with different Search Engine

Algorithm	Friedman
MTP	3.166667
BF	5.666667
Offline BF	5.958333
DE	6.208333
UMDA	6.208333
PESO	6.208333
PSO	6.458333
Offline FF	6.541667
FF	7.083333
Offline WF	9.083333
WF	9.083333
Offline AWF	10.500000
NF	11.166667
AWF	11.833333
Offline NF	14.833333

Table 6.3: Algorithm ranking based on a Friedman non-parametric test post-hoc procedure to discern the results obtained by Evolving and Reusing Heuristics

6.4 Results Analysis

It has been shown that is possible to generate one heuristic per instance set, this heuristic can solve well the instance set.

Once that the Grammar allow to sort the bins, it is possible to obtain a better results than the obtained by the FF because the Grammar try to fill the bins starting with the fuller.

All the heuristics generated by the GE can be applied to the test instances, unlike those generated by the GP [143] given that the 3% of all generated heuristic can not be applied, this is because the GE impose useful structures through the Grammar in BNF.

Chapter 7

Improving Heuristics

7.1 Introduction

In previous chapters has been shown that is possible to use the Grammar Evolution with Search Engines to evolve heuristics for the bin packing problem. Those heuristics have performance like the FirstFit and BestFit heuristic, this because the Grammars includes elements from those heuristics.

The methodology used in the present Chapter is the same that the used in Chapter 6, but in this Chapter is proposed a new Grammar to generate heuristics online or offline. Also was included new metaheuristics trying to obtain better results and with less parameters or with better exploitation or exploration methods.

The results obtained with the GE, using differents kernels, was compared against the results obtained by the classical heuristics, shown in Section 2.5 ,by using Friedman non-parametric test.

7.2 Methodology

The methodology used in the present Chapter is the same that the used in Chapter 6 , but in this Chapter is proposed a new Grammar to generate heuristics online or offline. Also was included new metaheuristics trying to obtain better results and with less parameters or with better exploitation or exploration methods.

The parameters used for each Search Engine are shown in Table 7.1. 33 experiments were performed independently and the median was used to compare the results against those obtained with the heuristics described in Section 2.5. The comparison was implemented through the non-parametric test of Friedman [141, 142] , this non-parametric test used a post-hoc analysis to discern the performance between the experiments and gives a ranking of them.

The Appendix A includes an Example to pack items based on classic heuristics and GE.

Instance	BSO	PSO	UMDA	cGA	PESO	DE
Functions Call	1500					
Population Size	50					
Strategie	-	-	-	-	-	DE/rand/1
F	-	-	-	-	-	0.9
C_r	-	-	-	-	-	0.8
w	1	1	-	-	1	-
ϕ_1	0.8	0.8	-	-	0.8	-
ϕ_2	0.5	0.5	-	-	0.5	-

Table 7.1: Parameters obtained by Covery Arrays to be used by GE to improve the heuristics

$$\begin{array}{rcl}
\langle \text{begin} \rangle & \models & \langle \text{exproff} \rangle \langle \text{exprsort} \rangle (\langle \text{expr} \rangle) <= (\langle \text{expr} \rangle) \\
\langle \text{exproff} \rangle & \models & Sort(Elements, \langle \text{order} \rangle) \mid \lambda \\
\langle \text{exprsort} \rangle & \models & Sort(\langle \text{exprkind} \rangle, \langle \text{order} \rangle) \mid \lambda \\
\langle \text{exprkind} \rangle & \models & \text{Bins} \mid \text{SumElements} \\
\langle \text{order} \rangle & \models & \text{Asc} \mid \text{Des} \\
\langle \text{expr} \rangle & \models & (\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle) \mid \langle \text{var} \rangle \mid abs(\langle \text{expr2} \rangle) \\
\langle \text{expr2} \rangle & \models & (\langle \text{expr2} \rangle \langle \text{op} \rangle \langle \text{expr2} \rangle) \mid \langle \text{var} \rangle \\
\langle \text{var} \rangle & \models & \text{F} \mid \text{C} \mid \text{S} \\
\langle \text{op} \rangle & \models & + \mid * \mid - \mid /
\end{array}$$

Grammar 7.1: Grammar proposal to generate heuristics online and offline, this grammar is based on Grammar 6.1

7.3 Results

The Table 7.2 shows an example of the heuristics obtained, there are shown only one example per instance set.

The Table 7.3 shows the results obtained with GE, it shows the results by each Search Engine. It was performed the Friedman non-parametric test to discern the results obtained, the Table 7.4 shows the ranking compute by the post-hoc procedure. The value of the non-parametric test is 104.51797 and the p-value 9.30069E-11.

Instance	Heuristic Generated
bindata1	Sort(Elements,Des).Sort(Bin,Des).((F+S))≤(abs(C))
bindata2	Sort(Elements,Des).Sort(Cont,Des).(abs(S))≤(abs((C-F))))
bindata3	Sort(Elements,Des).Sort(Cont,Des).(S)≤((C-F))
binpack1	Sort(Content,Des).(abs(F))≤((C-abs(S)))
binpack2	Sort(Content,Des).((F+S))≤(C)
binpack3	Sort(Content,Des).(F)≤(abs((C-S)))
binpack4	Sort(Content,Asc).(S)≤((C-F))
binpack5	((S+F))≤(C)
binpack6	(F)≤((abs(C)-S))
binpack7	(abs(F))≤(abs((S-C)))
binpack8	(abs((S+F)))≤(C)
hard28	Sort(Cont,Des).(F)≤(abs((C-S)))

Table 7.2: Example of the heuristic generate with the Grammar 7.1

Instance	BSO	PSO	UMDA	cGA	PESO	DE
bin1data	44.560112	44.561430	44.560112	44.561455	44.560112	44.560135
bin2data	44.560112	44.561430	44.560112	44.561455	44.560112	44.560135
bin3data	1.390289	1.390289	1.390289	1.390289	1.390289	1.390289
binpack1	0.913949	0.914034	0.913949	2.604965	0.913949	0.913949
binpack2	0.705970	0.706004	0.705970	2.396851	0.705970	0.705970
binpack3	0.591541	0.591543	0.591541	2.133326	0.591541	0.591541
binpack4	0.495512	0.495522	0.495512	1.935710	0.495512	0.495512
binpack5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
binpack6	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
binpack7	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
binpack8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
hard28	0.655480	0.655350	0.655480	0.655350	0.655480	0.655480

Table 7.3: Results obtained with the improving heuristics by using GE with different Search Engine

Algorithm	Friedman
MTP	3.666667
BSO	5.333333
UMDA	5.333333
PESO	5.333333
DE	5.833333
PSO	6.333333
BF	7.666667
Offline BF	7.875000
FF	8.333333
cGA	8.333333
Offline FF	9.125000
WF	10.750000
Offline WF	12.083333
NF	12.833333
Offline AWF	13.500000
AWF	13.833333
Offline NF	16.833333

Table 7.4: Algorithm ranking based on a Friedman non-parametric test post-hoc procedure to discern the results obtained by Improving Heuristics

7.4 Results Analysis

It have been shown that is possible to generate automatically heuristic through GE with better performance than the obtained by classic heuristic. The proposed Grammar allow to generate online and offline heuristics, it depends the kind of instance set used and the best way to handle it.

In the present Chapter were included the BSO and the cGA trying to obtain better results than the obtained with the others. The BSO is one of the metaheuristics that gives the best solutions, it is because the BSO includes Exploitation and Exploration that the PSO and BA doesn't includes. The cGA is UMDA's predecessor, however the simulation process that includes the cGA doesn't allow to diversify the solutions and the metaheuristic is trapped in local optima.

Chapter 8

Clustering Heuristics to reduce the heuristics generated

8.1 Introduction

The Chapter 5, 6 and 7 have shown that is possible to generate heuristics using GE, a Search Engine and a Grammar. The results obtained by the Grammar 7.1 shown better results than the obtained by the classic heuristics.

The aim of this chapter is to propose a way to reduce the number of heuristics generated. It was used Clustering process as metric to define the number of heuristics and assign a instances to a heuristics.

The results obtained by the GE with the Clustering process, using differents kernels, was compared against the results obtained by the classical heuristics, shown in Section 2.5 ,by using Friedman non-parametric test.

8.2 Clustering

The BPP doesn't have too much to parametrize apart the item's weight and the bin's capacity. Many researchers have investigated the range and weights and they have found that a smaller range makes the instance more difficult for common algorithms [144] [145] [146] [147].

Schwerin [146] showed that all the items inside the bin packing instance can be distributed as follows:

$$w \in [v_L\chi, v_U\chi] \tag{8.1}$$

where:

a) w: item.

b) v_L: lower bound.

c) v_U: upper bound.

d) χ: bin capacity.

The expression (8.1) normalizes the weights between 0 and 1. In [144] [148] was proposes to clustering the bin packing instances into three groups, as shown in Table 8.1, such clusters were based on the expression (8.1).

Group	Condition
Triplets	$v_L = 1/4$ y $v_U = 1/2$
Hards	$\overline{w} \simeq 1/3$ or close to $1/n$ where $n \geq 3$
Regulars	the others

Table 8.1: Bin Packing Instance Clusters

8.3 Methodology

The methodology used in this Chapter is based on the methodology shown in Chapter 7. One of the more important changes is that methodology generates no one heuristic per instance set, this methodology generate one heuristic per cluster. All the instances are join up and it generate only three heuristics using the Clustering shown in Section 8.2. With this only is necessary to test the instance with the HH and define if the heuristics has been evolved or if is necessary to define one heuristic.

33 experiments were performed independently and the median was used to compare the results against those obtained with the heuristics described in Section 2.5. The comparison was implemented through the non-parametric test of Friedman [141, 142] , this non-parametric test used a post-hoc analysis to discern the performance between the experiments and gives a ranking of them.

The Appendix A includes an Example to pack items based on classic heuristics and GE.

8.4 Results

The Table 8.2 shows an example of the heuristics obtained, there are shown only one example per Cluster.

It was performed the Friedman non-parametric test to discern the results obtained, the Table 8.3 shows the ranking compute by the post-hoc procedure. The value of the non-parametric test is 113.83615 and the p-value 7.49190E-11.

The Search Engines like BSO, UMDA and PESO obtained the best results with each Grammar. The comparison between the cluster process and without the cluster process was skipped because with a Friedman non-parametric test was shown that have the same performance.

Instance	Heuristic Generated
Triplets	$(\text{abs}((F + S))) \leq (C)$
Hard	$\text{Sort} (\text{Bin} , \text{Asc}) . ((S + F)) \leq (\text{abs}(C))$
Regular	$\text{Sort} (\text{Cont} , \text{Des}) . ((F + \text{abs}(S))) \leq (C)$

Table 8.2: Example of the heuristic generate with the Cluster

Algorithm	Friedman
MTP	5.333333
BSO with Grammar 7.1	7.166667
UMDA with Grammar 7.1	7.166667
PESO with Grammar 7.1	7.166667
DE with Grammar 7.1	7.666667
PSO with Grammar 7.1	8.333333
BF	10.750000
Offline BF	11.041667
DE with Grammar 6.1	11.875000
UMDA with Grammar 6.1	11.875000
PESO with Grammar 6.1	11.875000
Offline FF	12.458333
PSO with Grammar 6.1	12.458333
FF	13.000000
cGA with Grammar 5.1	13.000000
DE with Grammar 5.1	14.625000
PESO with Grammar 5.1	14.625000
UMDA with Grammar 5.1	14.625000
PSO with Grammar 5.1	14.791667
Offline WF	16.416667
WF	16.416667
Offline AWF	17.833333
NF	19.166667
AWF	20.833333
Offline NF	24.500000

Table 8.3: Algorithm ranking based on a Friedman non-parametric test post-hoc procedure to discern the results obtained by the clustering process and the different Search Engine with Grammars

8.5 Results Analysis

Based on the obtained results, it is possible to conclude that it is possible to cluster the instances using Schwerin's metric and generate heuristics using GE.

The results obtained show that with only three heuristics applied to the Bin Packing Instances we can obtain the same performance that if we make one heuristic by each instance set.

The algorithms PESO, BSO and UMDA got the best performance, among these the UMDA used the less parameters and the others need the Covery Arrays method to tuning them parameters. The worst a algorithms is the cGA due it is the UMDA's precursor.

This test included all the Grammars proposed with all the algorithms used, the Exact Method and the classic Heuristics, the aim is to compare the performances of these classic methods against those generated by the GE with the Search Engines and the differents Grammars.

It can be show how the MTP algorithm is the best way to solve a BPP instance, however due the BPP's complexity the heuristics generated by the GE using BSO, PESO or UMDA algorithms as Search Engine are the feasible way to solve the BPP instance.

Chapter 9

Conclusions and future work

This dissertation uses elements and methods at Optimization, Metaheuristic, Hyper-Heuristics and a Problem from Operations Research. The proposed approach allows to generate heuristics for a particular problem using Grammar Evolution, it was presented a generic framework to use differents fitness functions and Metaheuristics like an Hyper-Heuristics kernel.

The Chapter 5 presented a initial test to know if the GE can be used to generate Heuristics, this chapter is based on [18] where is used GP to evolve Bin Packing's heuristics. The Grammar 5.1 proposed in this chapter is based on the heuristics components shown in Table 5.1. The heuristics generated with this Grammar have a performance like the FF Heuristic.

The Chapter 6 shown a modified Grammar 6.1, this Grammar lets to the HH sort the bins to obtain a better performance that the Grammar 5.1. With this new Grammar was possible to generate heuristics who sorts bins,like the BF heuristic. The metaheuristics used were the same that in the Chapter 5.

In Chapter 7, was proposed the Grammar 7.1 to evolve heuristics online and offline, the aim of this Grammar is to choose the best way to pack items. Sometimes is better handle an instance with sort elements before to start the packing process, the HH using the Grammar can generate heuristics for both cases.

The Chapter 8 proposed a methodology to reduce the number of heuristic generated for the test instances. This methodology is based on clustering shown in Section 8.2, it allows to the HH generates only one heuristic per cluster. The results obtained shown that there aren't statistical difference between generate one heuristic per instances set or using the proposed methodology.

With the results obtained in the previous chapters, it can be concluded the follow:

a) It is possible to use GE like an improved of Genetic Programming, it is based on the GE uses a BNF Grammar to guide the evolution obtaining congruent solutions.

b) The GE can use different Search Engines to evolve the results however are prefer Search Engines who includes Exploration and Exploitation.

c) The parameters of the algorithms used into Search Engines must be tuning to obtain good results, it was used a fine tuning methodology based on Covery Arrays.

d) It is possible to generate a Grammar to evolve heuristics for a specific problem, the Grammar must include the problem's components.

e) It is possible to use the GE as Generation's HH.

f) It is possible to uses a Clustering Process to reduce the number of heuristics generated for the BPP, the obtained results are similar to the obtained without the Clustering Process but with less heuristics generated.

The main contributions of the present work are the follows:

a) To propose another metaheuristic to generate heuristics, like a generation's HH.

b) To incorporate different metaheuristics as kernel using the Grammar Evolution as an Indirect representation.

c) To propose grammars to evolve bin packing heuristics, the contributions of each one are the follows:

- ▷ The Grammar 5.1 generated heuristics with performance like the FF clasic heuristic.
- ▷ The Grammar 6.1 improved the heuristics generated by means the BF heuristic incorporating bin's sorts to obtain best results.
- ▷ The Grammar 7.1 handle the instance set like online or offline problem.

d) To generated by the GE can be applied to the BPP, unlike those generated by the GP [18] given that the 3% of all generated heuristic can not be applied, this is because the GE impose useful structures through the Grammar in the BNF.

There are some directions to research in order to extend the work presented in this thesis, Extensions and Future Work:

a) It is possible to apply another fine tuning algorithms, the Covering Arrays method used is based on a 3rd software made by the NIST however the construct of the Covery Arrays is a NP-Complete problem.

b) The problem shown in the present work is the 1D Bin Packing but it can extend to 2D and 3D Bin Packing problems, also can be used with Knapsack problems changing the Grammar.

c) The methodology presented can be applied to another kind of problems, is possible to generate heuristics to problems with unknown heuristics.

d) The heuristics generated contains a large number of terminals, with the BNF is not possible to reduce the number of terminals, it is necessary to look up a new Grammar that allows to reduce the heuristics generated.

Appendix A

Packing Examples

Suppose that we have the following instance:

a) c = 100

b) n = 10

c) w = 26, 26, 33, 20, 41, 49, 19, 34, 22, 29

A.1 MTP

The MTP can be used to pack the items, with this algorithm we obtain the following bins:

a) $[49, 29, 22]$

b) $[41, 33, 26]$

c) $[34, 26, 20, 19]$

When was applied the fitness functions, shown in Section 2.3, to the bins obtained by the MTP algorithm was obtained the following results: Function 2.6 = 3, Function 2.7 = 0.0100 and Function 2.8 = 0.0066. This algorithm is an Exact Method and the values obtained with it are the optimal results.

A.2 FirstFit

To solve the test instance using the online FF it is chosen each item and it is placed in the first bin that it can be placed. In this case it was begun with the item 26 and it was placed in the first bin. The second item, the item 26, is tested if it can be placed in the only bin used and the space occupied with this two elements doesn't exceed the bin size and the item is placed. The next item, the item 33, is tested again in the bin used previously and the three elements doesn't exceed the bin size and the item is placed. The next item, the item 20, is tested but with this element the bin exceeds its capacity and it is necessary to placed the item in a new bin. With the remaining elements is applied the same methodology and the bins obtained are the follow:

a) $[26, 26, 33]$

b) $[20, 41, 19]$

c) $[49, 34]$

d) $[22, 29]$

The results obtained with the fitness function are: Function 2.6 = 4, Function 2.7 = 1.01 and Function 2.8 = 0.4221.

The offline FF heuristic can be applied to the same instance, one of the most important changes is that the items must be ordering by the item weight. If we apply the offline FF then we have the items ordered as follows:

a) w = 49, 41, 34, 33, 29, 26, 26, 22, 20, 19

The way to pack them is the same of the online FF. The bins obtained by this heuristics are:

a) $[49, 41]$

b) $[34, 33, 29]$

c) $[26, 26, 22, 20]$

d) $[19]$

The results obtained with the fitness function are: Function 2.6 = 4, Function 2.7 = 1.01 and Function 2.8 = 0.3371.

The results obtained by the online and offline FF with the Fitness Function 2.6 and 2.7 are equals, however the Fitness Function 2.7 shown that is better to sort the items before to start the packing process.

A.3 BestFit

To solve the test instance using online BF is like the solve it with the FF but the BF sorts the bins from less to more space available and place the item in the first bin with space available, if not exists one bin with this condition is placed in a new bin. The items are placed as follows:

a) $[26, 26, 33]$

b) $[49, 34]$

c) $[20, 41, 19]$

d) $[22, 29]$

The results obtained with the fitness function are: Function 2.6 = 4, Function 2.7 = 1.01 and Function 2.8 = 0.4221.

The offline BF works like the online version, only needs to sort the items like the offline FF. The bins obtained by this heuristics are:

a) $[34, 33, 29]$

b) $[26, 26, 22, 20]$

c) $[49, 41]$

d) $[19]$

The results obtained with the fitness function are: Function 2.6 = 4, Function 2.7 = 1.01 and Function 2.8 = 0.3371.

A.4 Grammatical Evolution

To apply the GE is necessary to define the Search Engine and its parameters, however we gonna explain the process ones we have the Integer string, how it is mapping with the BNF and how it is testing with the instance.

Suppose that we have the next individuals of a certain metaheuristic, $ind_1 = \{1, 1, 2, 3, 1, 2, 3, 2, 1\}$, $ind_2 = \{1, 1, 2, 3, 1, 2, 2, 2, 1\}$ and $ind_3 = \{1, 1, 1, 2, 3, 2, 2, 2, 3, 3\}$, and we have the Grammar 5.1. To obtain the fitness of this individual is necessary to map the individual's values through the grammar, as shown in Figure 3.3. The mapping results are the follows:

a) $ind_1 = (S + S) \leq (F)$

b) $ind_2 = (S + C) \leq (F)$

c) $ind_3 = ((S * C)) - abs(< expr2 >)) \leq (< expr >)$

Now we can apply those heuristics to the instance, the way to apply this is to choose each item and test it with the heuristic generated.

The first generated heuristic is evaluated, trying to evaluate with each item and looking to comply the inequality, if the inequality is not meet then the item is place in a new bin. The bins obtained with this heuristics are:

a) $[26, 26, 20]$

b) $[33, 19, 22]$

c) $[41, 29]$

d) $[49]$

e) $[34]$

The results obtained with the fitness function and the heuristic $(S + S) \leq (F)$ are: Function 2.6 = 5, Function 2.7 = 2.01 and Function 2.8 = 0.6176.

The second heuristic is applied like the first, however the inequality never is meet, then all the items are placed in one bin and each bin only have one item. The bins obtained with this heuristics are:

a) $[26]$

b) $[26]$

c) $[20]$

d) $[33]$

e) $[19]$

f) $[22]$

g) $[41]$

h) $[29]$

i) $[49]$

j) $[34]$

The results obtained with the fitness function and the heuristic $(S + C) \leq (F)$ are: Function 2.6 = 10, Function 2.7 = 7.01 and Function 2.8 = 0.9023.

The last heuristic can't be applied because this heuristic have non-terminals and it can be replaced with values from the instance. In this case the fitness values are the infinity, with this the Search Engine will be discard those elements through the selection.

Glossary

AWF Almost Worst Fit. 13–16, 53, 59, 64, 70

BA Bee Algorithm. 29, 65

BF Best Fit. 13–16, 53, 55, 57, 59, 64, 70, 73, 75, 79

BNF Backus Naur Form grammar. 20, 21, 59, 74–76

BPP Bin Packing Problem. 7, 8, 11, 12, 54, 55, 67, 71, 74, 75

BSO Bee Swarm Optimization. 29, 62, 64, 65, 69–71

cGA Compact Genetic Algorithm. 31, 62, 64, 65, 70

DE Differential Evolution. 22, 24, 51, 53, 54, 58, 59, 62, 64, 70

EDA Estimation of Distribution Algorithm. 31

FF First Fit. 13–16, 53–55, 59, 64, 70, 73, 75, 78, 79

GA Genetic Algorithm. 17, 20, 22, 31

GE Grammatical Evolution. 20, 22, 49, 50, 52–59, 61–65, 67, 69, 71, 73–75

GP Genetic Programming. 17, 18, 20, 49, 59, 73, 75

HH Hyper-Heuristic. 33–43, 45–47, 55, 56, 69, 73, 74

MTP Martello and Toth Procedure. 11, 53, 59, 64, 70, 71, 77

NF Next Fit. 13–16, 53, 59, 64, 70

PESO Particle Evolutionary Swarm Optimization. 27, 51, 53, 54, 58, 59, 62, 64, 69–71

PSO Particle Swarm Optimization. 25–27, 29, 51, 53, 54, 58, 59, 62, 64, 65, 70

UMDA Univariate Marginal Distribution Algorithm. 31, 51, 53, 54, 58, 59, 62, 64, 65, 69–71

WF Worst Fit. 13–16, 53, 59, 64, 70

Bibliography

[1] S. Martello and P. Toth, *Knapsack Problems, Algorithms and and Computer Implementations.* New York, NY, USA: John Wiley & Sons Ltd., 1990.

[2] E. G. Coffman, Jr., C. Courcoubetis, M. R. Garey, D. S. Johnson, P. Shor, R. R. Weber, and M. Yannakakis, "Bin packing with discrete item sizes, part i: Perfect packing theorems and the average case behavior of optimal packings," *SIAM J. Disc. Math.*, vol. 13, pp. 384–402, 2000.

[3] T. G. Crainic, G. Perboli, M. Pezzuto, and R. Tadei, "New bin packing fast lower bounds," *Computers & Operations Research*, vol. 34, no. 11, pp. 3439 – 3457, 2007.

[4] E. Coffman, Jr., G. Galambos, S. Martello, and D. Vigo, *Bin Packing Approximation Algorithms: Combinatorial Analysis.* Kluwer Academic Publishers, 1998.

[5] J. Edward G. Coffman, G. Galambos, S. Martello, and D. Vigo, *Bin packing approximation algorithms: Combinatorial analysis*, pp. 151–207. Kluwer Academic Pub., 1999.

[6] S. P. Fekete and J. Schepers, "New classes of fast lower bounds for bin packing problems," *Mathematical Programming*, vol. 91, no. 1, pp. 11–31, 2001.

[7] S. S. Seiden, R. van Stee, and L. Epstein, “New bounds for variable-sized online bin packing,” *SIAM Journal on Computing*, vol. 32, p. 2003, 2003.

[8] E. G. Coffman Jr. and J. Csirik, “A classification scheme for bin packing theory,” *Acta Cybernetica*, vol. 18, pp. 47–60, 2007.

[9] T. Kämpke, “Simulated annealing: Use of a new tool in bin packing,” *Annals of Operations Research*, vol. 16, pp. 327–332, 1988.

[10] E. Falkenauer and A. Delchambre, “A genetic algorithm for bin packing and line balancing,” in *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pp. 1186 –1192 vol.2, may 1992.

[11] A. Lodi, S. Martello, and D. Vigo, “Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems,” *INFORMS Journal on Computing*, vol. 11, no. 4, pp. 345–357, 1999.

[12] E. Hopper and B. Turton, “A review of the application of meta-heuristic algorithms to 2d strip packing problems,” *Artificial Intelligence Review*, vol. 16, no. 4, pp. 257–300, 2001.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co., 1979.

[14] J. Beasley, “Or-library: distributing test problems by electronic mail,” *Journal of the Operational Research Society*, vol. 41, no. 11, pp. 1069–1072, 1990.

[15] E. Falkenauer, “A hybrid grouping genetic algorithm for bin packing,” *Journal of Heuristics*, vol. 2, pp. 5–30, 1996.

[16] A. Scholl, R. Klein, and C. Jürgens, “Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem,” *Computers & Operations Research*, vol. 24, no. 7, pp. 627 – 645, 1997.

[17] A. Alvim, C. Ribeiro, F. Glover, and D. Aloise, “A hybrid improvement heuristic for the one-dimensional bin packing problem,” *Journal of Heuristics*, vol. 10, no. 2, pp. 205–229, 2004.

[18] M. Hyde, *A Genetic Programming Hyper-Heuristic Approach to Automated Packing.* PhD thesis, University of Nottingham, 2010.

[19] W. T. Rhee and M. Talagrand, “On line bin packing with items of random size,” *Mathematics of Operations Research*, vol. 18, no. 2, pp. 438–445, 1993.

[20] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, “Worst-case performance bounds for simple one-dimensional packing algorithms,” *SIAM Journal on Computing*, vol. 3, no. 4, pp. 299–325, 1974.

[21] A. C.-C. Yao, “New algorithms for bin packing,” *J. ACM*, vol. 27, pp. 207–227, April 1980.

[22] K. Fleszar and K. S. Hindi, “New heuristics for one-dimensional bin-packing,” *Computers & Operations Research*, vol. 29, no. 7, pp. 821 – 839, 2002.

[23] C. Toledo Suarez, E. Gonzlez, and M. Rendon, “A heuristic algorithm for the offline one-dimensional bin packing problem inspired by the point jacobi matrix iterative method,” in *Artificial Intelligence, 2006. MICAI ’06. Fifth Mexican International Conference on*, pp. 281–286, Nov 2006.

[24] S.-C. Tam, H.-K. Tam, L.-M. Tam, and T. Zhang, “A new optimization method, the algorithm of changes, for bin packing problem,” in *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*, pp. 994–999, Sept 2010.

[25] J. R. Koza, “Hierarchical genetic algorithms operating on populations of computer programs,” in *IJCAI*, pp. 768–774, 1989.

[26] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection.* Cambridge, MA: The MIT Press, 1992.

[27] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming An Introduction; On the Automatic Evolution of Computer Programs and its Applications.* San Francisco, CA, USA: Morgan Kaufmann, Jan. 1998.

[28] J. R. Koza and R. Poli, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, ch. Genetic programming, pp. 127–164. Springer, May 2006.

[29] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proceedings of the 1st International Conference on Genetic Algorithms*, (Hillsdale, NJ, USA), pp. 183–187, L. Erlbaum Associates Inc., 1985.

[30] C. Ryan, J. Collins, J. Collins, and M. O'Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming*, pp. 83–95, Springer-Verlag, 1998.

[31] J. R. Koza and R. Poli, "Genetic programming," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (E. K. Burke and G. Kendall, eds.), pp. 127–164, Boston: Kluwer, 2005.

[32] I. Dempsey, M. O'Neill, and A. Brabazon, "Foundations in grammatical," in *Foundations in Grammatical Evolution for Dynamic Environments*, vol. 194, New York, NY, USA: Springer-Verlag, 2009.

[33] H. lan Fang, H. lan Fang, P. Ross, P. Ross, D. Corne, and D. Corne, "A promising genetic algorithm approach to job-shop scheduling, rescheduling,

and open-shop scheduling problems," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 375–382, Morgan Kaufmann, 1993.

[34] J. Holland, "Adaptation in natural and artificial systems," *University of Michigan Press*, 1975.

[35] O. M. and B. A, "Grammatical differential evolution," in *International Conference on Artificial Intelligence (ICAI'06)*, (Las Vegas, Nevada), CSEA Press, 2006.

[36] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *J. of Global Optimization*, vol. 11, pp. 341–359, December 1997.

[37] X. S. Yang, *Nature Inspired Metaheuristic Algorithms.* Luniver Press, 2da ed., 2008.

[38] E. n. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello, "A comparative study of differential evolution variants for global optimization," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, (New York, NY, USA), pp. 485–492, ACM, 2006.

[39] A. Qin, V. Huang, and P. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 32, pp. 398 – 417, April 2009.

[40] S. Luke, *Essentials of Metaheuristics.* Lulu, 2009.

[41] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," *IEEE Int. Conf. Neural Netw*, vol. 4, pp. 1942–1948, 1995.

[42] C. Maurice, *Particle Swarm Optimization.* Estados Unidos: Wiley-ISTE, 2006.

[43] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, pp. 33–57, June 2007.

[44] M. F. Tasgetiren, P. N. Suganthan, and Q.-Q. Pan, "A discrete particle swarm optimization algorithm for the generalized traveling salesman problem," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, (New York, NY, USA), pp. 158–167, ACM, 2007.

[45] T. Gong and A. L. Tuson, "Binary particle swarm optimization: a forma analysis approach," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, (New York, NY, USA), pp. 172–172, ACM, 2007.

[46] A. E. Muñoz Zavala, A. H. Aguirre, and E. R. Villa Diharce, "Constrained optimization via particle evolutionary swarm optimization algorithm (peso)," in *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, GECCO '05, (New York, NY, USA), pp. 209–216, ACM, 2005.

[47] A. Zavala, A. Aguirre, and E. Diharce, "Particle evolutionary swarm optimization algorithm (peso)," in *Computer Science, 2005. ENC 2005. Sixth Mexican International Conference on*, pp. 282–289, 2005.

[48] A. Munoz-Zavala, A. Hernandez-Aguirre, E. Villa-Diharce, and S. Botello-Rionda, "Peso+for constrained optimization," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pp. 231–238, 2006.

[49] G. Pulido and C. Coello, "A constraint-handling mechanism for particle swarm optimization," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 2, pp. 1396–1403 Vol.2, 2004.

[50] M. A. Sotelo-Figueroa, M. del Rosario Baltazar-Flores, and J. M. Carpio, "Application of bee swarm optimization "bso" to the knapsack problem," in

International Seminar on Computational Intelligence 2010, Springer-Verlag, 2010.

[51] B. Xing and W.-J. Gao, "Bee inspired algorithms," in *Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms*, vol. 62 of *Intelligent Systems Reference Library*, pp. 45–80, Springer International Publishing, 2014.

[52] D. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi, "The bees algorithm - a novel tool for complex optimization problems," in *Intelligent Production Machines and Systems*, pp. pp. 454–460, Elsevier, 2006.

[53] G. Harik, F. Lobo, and D. Goldberg, "The compact genetic algorithm," *Evolutionary Computation, IEEE Transactions on*, vol. 3, pp. 287–297, Nov 1999.

[54] J. Gallagher, S. Vigraham, and G. Kramer, "A family of compact genetic algorithms for intrinsic evolvable hardware," *Evolutionary Computation, IEEE Transactions on*, vol. 8, pp. 111–126, April 2004.

[55] R. Rastegar and A. Hariri, "A step forward in studying the compact genetic algorithm," *Evolutionary Computation*, vol. 14, pp. 277–289, 2014/11/08 2006.

[56] H. Mühlenbein and G. Paaß, "From recombination of genes to the estimation of distributions i. binary parameters," in *Parallel Problem Solving from Nature — PPSN IV* (H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), vol. 1141 of *Lecture Notes in Computer Science*, pp. 178–187, Springer Berlin Heidelberg, 1996.

[57] P. Larrañaga, *Estimation of distribution algorithms a new tool for evolution computation.* Norwell, MA, USA: Kluwer Academic Pub., 2002.

[58] M. Pelikan, D. Goldberg, and F. Lobo, "A survey of optimization by building and using probabilistic models," *Computational Optimization and Applications*, vol. 21, no. 1, pp. 5–20, 2002.

[59] E. K. Burke, B. L. MacCarthy, S. Petrovic, and R. Qu, "Multiple-retrieval case-based reasoning for course timetabling problems," *Journal of the Operational Research Society*, vol. 57, no. 2, pp. 148–162, 2006.

[60] H. Fisher and G. L. Thompson, "Probabilistic learning combinations of local job-shop scheduling rules," in *Factory Scheduling Conference*, 1961.

[61] H. Fisher and G. L. Thompson, *Probabilistic learning combinations of local job-shop scheduling rules*, pp. 225–251. New Jersey: Prentice-Hall, Inc, 1963.

[62] A. Caprara and M. Monaci, "On the two-dimensional knapsack problem," *Operations Research Letters*, vol. 32, no. 1, pp. 5 – 14, 2004.

[63] D. Whitley and J. Watson, "Complexity theory and the no free lunch theorem," in *Search Methodologies* (E. K. Burke and G. Kendall, eds.), vol. 1, pp. 317–339, Springer US, 2005.

[64] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 67–82, Apr. 1997.

[65] E. Burke, Y. Bykov, J. Newall, and S. Petrovic, "A time-predefined approach to course timetabling," *Yugoslav Journal of Operations Research ISSN: 0354-0243 EISSN: 2334-6043*, vol. 13, no. 2, 2003.

[66] E. K. Burke, Y. Bykov, J. Newall, and S. Petrovic, "A time-predefined local search approach to exam timetabling problems," *IIE Transactions*, vol. 36, no. 6, pp. 509–528, 2004.

[67] E. K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenbur, "Hyperheuristics: An emerging direction in modern search technology," in *Handbook of Meta-Heuristics* (F. Glover and G. Kochenberger, eds.), pp. 457–474, Kluwer, 2003.

[68] P. Ross, S. Schulenbur, J. G. Marín-Blázquez, and E. Hart, "Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics," *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1295–1306, 2003.

[69] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT 2000)* (E. K. Burke and W. Erben, eds.), pp. 176–190, Aug. 2000.

[70] S. Summit, P. Cowling, G. Kendall, and E. Soubeiga, "A parameter-free hyperheuristic for scheduling a sales summit," in *4th International Conference*, pp. 127–131, July 2001.

[71] P. Cowling, G. Kendall, and E. Soubeiga, "Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation," in *Applications of Evolutionary Computing* (S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. Raidl, eds.), vol. 2279 of *Lecture Notes in Computer Science*, pp. 269–287, Springer Berlin / Heidelberg, 2002. 10.1007/3-540-46004-7_1.

[72] P. Cowling, G. Kendall, and L. Han, "An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem," tech. rep., University of Nottingham, UK, 2002.

[73] E. Soubeiga, *Development and application of hyperheuristics to personnel scheduling.* PhD thesis, University of Nottingham, 2003.

[74] F. Glover and G. A. Kochenberger, *Handbook of metaheuristics.* Kluver Academic Publishers, 2003.

[75] E. K. Burke, G. Kendall, and E. Soubeiga, "A tabu-search hyper-heuristic for timetabling and rostering," *Journal of Heuristics*, vol. 9, no. 6, pp. 451–470, 2003.

[76] G. Kendall and N. Hussin, "An investigation of a tabu-search-based hyper-heuristic for examination timetabling," in *Multidisciplinary Scheduling: Theory and Applications* (G. Kendall, E. K. Burke, S. Petrovic, and M. Gendreau, eds.), pp. 309–328, Springer US, 2005.

[77] G. Kendall and N. Hussin, "A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology," in *Practice and Theory of Automated Timetabling V* (E. Burke and M. Trick, eds.), vol. 3616 of *Lecture Notes in Computer Science*, pp. 270–293, Springer Berlin / Heidelberg, 2005.

[78] P. Rattadilok, A. Gaw, and R. Kwan, "Distributed choice function hyper-heuristics for timetabling and scheduling," in *Practice and Theory of Automated Timetabling V* (E. Burke and M. Trick, eds.), vol. 3616 of *Lecture Notes in Computer Science*, pp. 51–67, Springer Berlin Heidelberg, 2005.

[79] P. Ross, S. Schulenbur, J. G. Marín-Blázquez, and E. Hart, "Hyper-heuristics: learning to combine simple heuristics in bin-packing problems," *Genetic and Evolutionary Computation Conference (GECCO)*, 2002.

[80] H. Terashima-Marín, E. J. Flores-Álvarez, and P. Ross, "Hyper-heuristics and classifier systems for solving 2d-regular cutting stock problems," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, (New York, NY, USA), pp. 637–643, ACM, 2005.

[81] H. Terashima-Marín, C. Farías-Zárate, P. Ross, and M. Valenzuela-Rendón, "A ga-based method to produce generalized hyper-heuristics for the 2d-regular cutting stock problem," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, (New York, NY, USA), pp. 591–598, ACM, 2006.

[82] H. Terashima-Marin, C. J. Farias Zarate, P. Ross, and M. Valenzuela-Rendon, "Comparing two models to generate hyper-heuristics for the 2d-regular bin-packing problem," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, vol. 1 of *GECCO '07*, (New York, NY, USA), pp. 2182–2189, ACM, 2007.

[83] L. Han and G. Kendall, "Guided operators for a hyper-heuristic genetic algorithm," in *IN PROCEEDINGS OF AI-2003: ADVANCES IN ARTIFICIAL INTELLIGENCE. THE 16TH AUSTRALIAN CONFERENCE ON ARTIFICIAL INTELLIGENCE (AI'03)*, pp. 807–820, 2003.

[84] L. Han and G. Kendall, "An investigation of a tabu assisted hyper-heuristic genetic algorithm," in *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, vol. 3, pp. 2230 – 2237 Vol.3, dec. 2003.

[85] L. Han, G. Kendall, and P. Cowling, "An adaptive length chromosome hyper-heuristic genetic algorithm for a trainer scheduling problem," in *Proceedings of the fourth Asia-Pacific Conference on Simulated Evolution And Learning, (SEAL'02), Orchid Country Club, Singapore, 18-22 Nov 2002*, vol. 1, (Singapore), pp. 267–271, November 2002.

[86] E. Burke, G. Kendall, D. L. Silva, R. O'Brien, D. L, A. Silva, and E. Soubeiga, "An ant algorithm hyperheuristic for the project presentation scheduling problem," in *In: Proceedings of the Congress on Evolutionary Computation 2005 (CEC'05). Volume 3*, vol. 1, pp. 2263–2270, IEEE press, 2005.

[87] A. Cuesta-Cañada, L. Garrido, and H. Terashima-Marín, "Building hyper-heuristics through ant colony optimization for the 2d bin packing problem," in *Knowledge-Based Intelligent Information and Engineering Systems* (R. Khosla, R. J. Howlett, and L. C. Jain, eds.), vol. 3684 of *Lecture Notes in Computer Science*, pp. 654–660, Springer Berlin / Heidelberg, 2005. 10.1007/11554028_91.

[88] K. A. Dowsland, E. Soubeiga, and E. Burke, "A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation," *European Journal of Operational Research*, vol. 179, no. 3, pp. 759–774, 2007.

[89] R. Bai and G. Kendall, "An investigation of automated planograms using a simulated annealing based hyper-heuristics," in *Metaheuristics: Progress as Real Problem Solvers* (T. Ibaraki, K. Nonobe, and M. Yagiura, eds.), vol. 32 of *Operations Research/Computer Science Interfaces*, pp. 87–108, Springer Berlin / Heidelberg, 2005.

[90] Bai, R, Burke, E K, Kendall, and G, "Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation," *Journal of the Operational Research Society*, vol. 59, pp. 1387–1397, Oct. 2008.

[91] R. Bai, E. Burke, M. Gendreau, G. Kendall, and B. McCollum, "Memory length in hyper-heuristics: An empirical study," in *Computational Intelligence in Scheduling, 2007. SCIS '07. IEEE Symposium on*, vol. 1, pp. 173 –178, april 2007.

[92] E. Özcan, B. Bilgin, and E. Korkmaz, "Hill Climbers and Mutational Heuristics in Hyperheuristics," in *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006)* (T. Runarsson, H.-G.

Beyer, E. Burke, J. J.Merelo-Guervos, D. Whitley, and X. Y. E. K, eds.), pp. 202–211, LNCS 4193, 2006.

[93] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, “A graph-based hyper heuristic for timetabling problems,” *European Journal of Operational Research*, vol. 176, pp. 177–192, 2007.

[94] H. Terashima-Marín and P. Ross, “Evolution of constraint satisfaction strategies in examination timetabling,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99)*, pp. 635–642, Morgan Kaufmann, 1999.

[95] P. Ross, E. Hart, and D. Corne, “Some observations about ga-based exam timetabling,” in *Selected papers from the Second International Conference on Practice and Theory of Automated Timetabling II*, (London, UK), pp. 115–129, Springer-Verlag, 1998.

[96] B. Bilgin, E. Özcan, and E. E. Korkmaz, “An experimental study on hyper-heuristics and exam timetabling,” in *Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling (PATAT’06)*, pp. 123–140, August 2006.

[97] E. Ersoy, E. Ozcan, and S. Uyar, “Memetic algorithms and hyperhill-climbers,” in *Proceedings of the 3rd Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA’07)*, (Paris, France), pp. 159–166, August 2007.

[98] U. Dorndorf and E. Pesch, “Evolution based learning in a job shop scheduling environment,” *Comput. Oper. Res.*, vol. 22, pp. 25–40, January 1995.

[99] R. H. Storer, S. D. Wu, and R. Vaccari, “New search spaces for sequencing problems with application to job shop scheduling,” *Manage. Sci.*, vol. 38, pp. 1495–1509, October 1992.

[100] R. H. Storer, S. D. Wu, and R. Vaccari, “Problem and heuristic space search strategies for job shop scheduling,” *ORSA Journal on Computing*, vol. 7, no. 4, pp. 453–467, 1995.

[101] E. Hart and P. Ross, “A heuristic combination method for solving job-shop scheduling problems,” in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, PPSN V, (London, UK), pp. 845–854, Springer-Verlag, 1998.

[102] H. lan Fang, P. Ross, P. Ross, D. Corne, and D. Corne, “A promising hybrid ga/heuristic approach for open-shop scheduling problems,” in *ECAI 94 Proceedings of the 11th European Conference on Artificial Intelligence*, pp. 590–594, John Wiley and Sons, Ltd, 1994.

[103] P. Cowling and K. Chakhlevitch, “Hyperheuristics for managing a large collection of low level heuristics to schedule personnel,” in *Evolutionary Computation, 2003. CEC ’03. The 2003 Congress on*, vol. 2, pp. 1214 – 1221 Vol.2, dec. 2003.

[104] K. Chakhlevitch and P. Cowling, “Choosing the fittest subset of low level heuristics in a hyperheuristic framework,” in *Evolutionary Computation in Combinatorial Optimization* (G. Raidl and J. Gottlieb, eds.), vol. 3448 of *Lecture Notes in Computer Science*, pp. 23–33, Springer Berlin / Heidelberg, 2005.

[105] E. K. Burke, S. Petrovic, and R. Qu, “Case-based heuristic selection for timetabling problems,” *J. of Scheduling*, vol. 9, no. 2, pp. 115–132, 2006.

[106] E. Burke, M. Dror, S. Petrovic, and R. Qu, "Hybrid graph heuristics within a hyper-heuristic approach to exam timetabling problems," in *The Next Wave in Computing, Optimization, and Decision Technologies* (B. Golden, S. Raghavan, E. Wasil, R. Sharda, and S. Voß, eds.), vol. 29 of *Operations Research/-Computer Science Interfaces Series*, pp. 79–91, Springer US, 2005.

[107] J. Gratch and G. D. Composer, "A probabilistic solution to the utility problem in speed-up learning," in *10th National Conference on Artificial Intelligence (AAAI'92)*, (San Jose, California), pp. 235–240, July 1992.

[108] J. Gratch and S. Chien, "Learning search control knowledge for the deep space network scheduling problem," tech. rep., Champaign, IL, USA, 1993.

[109] J. Gratch and S. Chien, "Adaptive problem-solving for large-scale scheduling problems: a case study," *J. Artif. Int. Res.*, vol. 4, pp. 365–396, May 1996.

[110] J. McDermott, "Preliminary steps toward a taxonomy of problem-solving methods," in *Automating knowledge acquisition for expert systems*, pp. 225–256, Springer, 1988.

[111] J. Egeblad and D. Pisinger, "Heuristic approaches for the two-and three-dimensional knapsack packing problem," *Computers & Operations Research*, vol. 36, no. 4, pp. 1026–1049, 2009.

[112] P. Hansen and N. Mladenovic, "An introduction to variable neighborhood search," in *Metaheuristics, advances and trends in local search paradigms for optimization* (S. Voss, I. H. Osman, and C. Roucairol, eds.), (Norwell, MA, USA), pp. 433–458, Kluwer, 1999.

[113] B. Wah, "Population-based learning: a method for learning from examples under resource constraints," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 4, pp. 454 –474, oct 1992.

[114] B. Wah, A. Ieumwananonthachai, L.-C. Chu, and A. Aizawa, "Genetics-based learning of new heuristics: rational scheduling of experiments and generalization," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 7, pp. 763 –785, oct 1995.

[115] A. Ieumwananonthachai, *Automated Design of Knowledge-Lean Heuristics: Learning, Resource Scheduling, and Generalization.* PhD thesis, Dept. of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, 1996.

[116] Wah, , B. W. Wah, and A. Ieumwananonthachai, "Teacher: A genetics-based system for learning and for generalizing heuristics," in *Evolutionary Computation. World Scientific Publishing Co. Pte. Ltd*, pp. 179–211, Springer-Verlag, 1998.

[117] A. Ieumwananonthachai, A. N. Aizawa, S. R. Schwartz, B. W. Wah, and J. C. Yan, "Intelligent process mapping through systematic improvement of heuristics," *J. Parallel Distrib. Comput.*, vol. 15, pp. 118–142, June 1992.

[118] C.-C. Teng and B. Wah, "An automated design system for finding the minimal configuration of a feed-forward neural network," in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, vol. 3, pp. 1295 –1300 vol.3, jun-2 jul 1994.

[119] N. Krasnogor., *Studies on the Theory and Design Space of Memetic Algorithms.* PhD thesis, University of the West of England, 2002.

[120] C. Kenyon, "Best-fit bin-packing with random order," in *In 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, vol. 1, pp. 359–364, 1997.

[121] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward, "Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a

master of one," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, (New York, NY, USA), pp. 1559–1565, ACM, 2007.

[122] *A histogram-matching approach to the evolution of bin-packing strategies*, 2007.

[123] A. Fukunaga, "Automated discovery of composite sat variable-selection heuristics," in *Eighteenth national conference on Artificial intelligence*, vol. 1, (Menlo Park, CA, USA), pp. 641–648, American Association for Artificial Intelligence, 2002.

[124] A. S. Fukunaga, "Evolving local search heuristics for sat using genetic programming," in *Genetic and Evolutionary Computation – GECCO 2004*, vol. 3103 of *Lecture Notes in Computer Science*, pp. 483–494, Springer Berlin - Heidelberg, 2004.

[125] M. Bader-El-Den and R. Poli, "Generating sat local-search heuristics using a gp hyper-heuristic framework," in *Artificial Evolution* (N. Monmarché, E.-G. Talbi, P. Collet, M. Schoenauer, and E. Lutton, eds.), vol. 4926 of *Lecture Notes in Computer Science*, pp. 37–49, Springer Berlin / Heidelberg, 2008.

[126] M. El Den and R. Poli, "Grammar-based genetic programming for timetabling," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pp. 2532 –2539, may 2009.

[127] R. Keller and R. Poli, "Linear genetic programming of parsimonious metaheuristics," in *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pp. 4508 –4515, sept. 2007.

[128] N. B. Ho and J. C. Tay, "Evolving dispatching rules for solving the flexible job-shop problem," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3, pp. 2848 – 2855 Vol. 3, sept. 2005.

[129] J. C. Tay and N. B. Ho, "Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems," *Comput. Ind. Eng.*, vol. 54, pp. 453–473, April 2008.

[130] D. Jakobovic, L. Jelenkovic, and L. Budin, "Genetic programming heuristics for multiple machine scheduling," in *Proceedings of the 10th European conference on Genetic programming*, EuroGP'07, (Berlin, Heidelberg), pp. 321–330, Springer-Verlag, 2007.

[131] C. Dimopoulos and A. Zalzala, "Investigating the use of genetic programming for a classic one-machine scheduling problem," *Advances in Engineering Software*, vol. 32, no. 6, pp. 489 – 498, 2001.

[132] C. D. Geiger, R. Uzsoy, and H. Aytug, "Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach," *J. of Scheduling*, vol. 9, pp. 7–34, February 2006.

[133] R. Kumar, A. H. Joshi, K. K. Banka, and P. I. Rockett, "Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, (New York, NY, USA), pp. 1227–1234, ACM, 2008.

[134] R. Kumar, B. K. Bal, and P. I. Rockett, "Multiobjective genetic programming approach to evolving heuristics for the bounded diameter minimum spanning tree problem: Mogp for bdmst," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, (New York, NY, USA), pp. 309–316, ACM, 2009.

[135] M. Oltean, "Evolving evolutionary algorithms using linear genetic programming," *Evol. Comput.*, vol. 13, pp. 387–410, September 2005.

[136] J. Tavares, P. Machado, A. Cardoso, F. Pereira, and E. Costa, "On the evolution of evolutionary algorithms," in *Genetic Programming* (M. Keijzer, U.-M. O'Reilly, S. Lucas, E. Costa, and T. Soule, eds.), vol. 3003 of *Lecture Notes in Computer Science*, pp. 389–398, Springer Berlin / Heidelberg, 2004.

[137] G. L. Pappa and A. A. Freitas, "Automatically evolving rule induction algorithms tailored to the prediction of postsynaptic activity in proteins," *Intelligent Data Analysis*, vol. 13, no. 2, pp. 243–259, 2009.

[138] N. Krasnogor, "Self generating metaheuristics in bioinformatics: The proteins structure comparison case," *Genetic Programming and Evolvable Machines*, vol. 5, pp. 181–201, June 2004.

[139] N. Krasnogor and S. Gustafson, "A study on the use of "self-generation" in memetic algorithms," *Natural Computing*, vol. 3, no. 1, pp. 53—76, 2003.

[140] M. R. Hyde, E. K. Burke, and G. Kendall, "Automated code generation by local search," *J Oper Res Soc*, vol. 64, pp. 1725–1741, Dec 2013.

[141] J. Derrac, S. García, S. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, pp. 3–18, 2011.

[142] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures.* CRC, 2nd. ed., 2000.

[143] E. K. Burke, M. Hyde, and G. Kendall, "Evolving bin packing heuristics with genetic programming," in *Parallel Problem Solving from Nature - PPSN IX* (T. Runarsson, H.-G. Beyer, E. Burke, J. Merelo-Guervós, L. Whitley, and X. Yao, eds.), vol. 4193 of *Lecture Notes in Computer Science*, pp. 860–869, Springer Berlin / Heidelberg, 2006.

[144] E. Falkenauer, “Tapping the full power of genetic algorithm through suitable representation and local optimization: Application to bin packing,” in *Evolutionary Algorithms in Management Applications* (J. Biethahn and V. Nissen, eds.), pp. 167–182, Springer Berlin Heidelberg, 1995.

[145] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, “Approximation algorithms for np-hard problems,” ch. Approximation Algorithms for Bin Packing: A Survey, pp. 46–93, Boston, MA, USA: PWS Publishing Co., 1997.

[146] P. Schwerin and G. Wäscher, “The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp,” *International Transactions in Operational Research*, vol. 4, no. 5-6, pp. 377–389, 1997.

[147] I. Gent, “Heuristic solution of open bin packing problems,” *Journal of Heuristics*, vol. 3, no. 4, pp. 299–304, 1998.

[148] K. Smith-Miles and L. Lopes, “Measuring instance difficulty for combinatorial optimization problems,” *Computers & Operations Research*, vol. 39, pp. 875–889, May 2012.