Evolutionary Algorithms and Hyper-heuristics for Orthogonal Packing Problems

Qiang Guo

Thesis submitted to The University of Nottingham
for the degree of Doctor of Philosophy

March 2011

PAGINATED BLANK PAGE SCANNED AS FOUND IN ORIGINAL THESIS

NO INFORMATION IS MISSING

PAGE NUMBERING AS ORIGINAL

Abstract

This thesis investigates two major classes of Evolutionary Algorithms, Genetic Algorithms (GAs) and Evolution Strategies (ESs), and their application to the Orthogonal Packing Problems (OPP). OPP are canonical models for NP-hard problems, the class of problems widely conceived to be unsolvable on a polynomial deterministic Turing machine, although they underlie many optimisation problems in the real world. With the increasing power of modern computers, GAs and ESs have been developed in the past decades to provide high quality solutions for a wide range of optimisation and learning problems. These algorithms are inspired by Darwinian nature selection mechanism that iteratively select better solutions in populations derived from recombining and mutating existing solutions. The algorithms have gained huge success in many areas, however, being stochastic processes, the algorithms' behaviour on different problems is still far from being fully understood. The work of this thesis provides insights to better understand both the algorithms and the problems.

The thesis begins with an investigation of hyper-heuristics as a more general search paradigm based on standard EAs. Hyper-heuristics are shown to be able to overcome the difficulty of many standard approaches which only search in partial solution space. The thesis also looks into the fundamental theory of GAs, the schemata theorem and the building block hypothesis, by developing the Grouping Genetic Algorithms (GGA) for high dimensional problems and providing supportive yet qualified empirical evidences for the hypothesis. Realising the difficulties of genetic encoding over combinatorial search domains, the thesis proposes a phenotype representation together with Evolution Strategies that operates on such representation. ESs were previously applied mainly to continuous numerical optimisation, therefore being less understood when searching in combinatorial domains. The work in this thesis develops highly competent ES algorithms for OPP and opens the door for future research in this area.

Acknowledgements

The study on the area of combinatorial optimisation and evolutionary algorithms is a challenging but exciting journey. Without the help from many people it would not be possible to finish the journey. Firstly I would like to thank my PhD supervisors, Professor Edmund Burke and Professor Graham Kendall. They provided me the wonderful opportunity to study in the area, and always have good ideas whenever I encounter problems. They are also extremely patient with my progress and let me explore interesting ideas freely. I also thank Professor Chris Potts and Professor Miguel Anjos for their inspiring lectures during my Master's study. I am indebted to Professor Jay Yellen at Rollins College. During his visit to my school, we discussed some ideas on graph modelling for packing problems. He gave me almost one-on-one tutorials on graph theory which is a very important foundation in combinatorial optimisation and will greatly help my future research. Lastly I thank my family for their constant support and unconditional love.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Orthogonal Packing Problems (OPP) belong to the large family of cutting and packing problems [60, 178], where the objective is to optimise the arrangement of items into a number of containers. It fits into the paradigm of optimisation, that minimise (or maximise) an objective function (such as the cost of material), subject to a set of other constraints (such as limit of weights or sizes). The problem arises in many industrial settings, from stock-cutting in paper, metal, glass and many other raw material industries to container, pallet loading in logistic industries, or even multi-processor scheduling, portfolio management and other resource allocation problems. Though having a wide variety of applications, the problem is very difficult to solve, except for small-sized instances which may be solvable by exact methods. Underlying these applications is the fact that they are NP-hard problems [79]. NP-hard problems are a class of problems that are widely believed not to be solvable in polynomial time on a deterministic Turing machine (not to mention limited memory size in reality). The problem has been receiving attention from many disciplines for a long time. From an operational research (or management science) perspective, it remains an extremely challenging topic in

optimization problems and complexity theory.

One of the traditional ways for tackling these problems is by exact methods such as Mathematical Programming [81, 82], and Dynamic Programming [118, 175]. These methods can guarantee optimality but only for small-sized instances. Another approach is to use heuristics which usually gives fast, feasible results even for large sized problems but without the guarantee of global optimality. For some sub-classes of NP-hard problems, there are some heuristics (Polynomial Time Approximation Scheme (PTAS)) that can produce approximate solutions within a constant factor to the optima [64]. However, there are also limitations that many sub-classes of problems have not had any suitable approximation algorithms [180]; or for those problems having such PTAS, there may exist an optimal approximation ratio which imposes a gap to optima [123] that cannot be overcome. More recently Evolutionary Algorithms (EAs), among other meta-heuristics, has been a fruitful research area for Combinatorial Optimisation Problems thanks to the increasing computational power of modern computers. Meta-heuristics are often inspired by natural phenomena and work amazingly well across a broad categories of problems. EAs adopt a more stochastic search paradigm and normally consists of the mechanisms of recombination, mutation and selection on a population of candidate solutions. However, EAs are still far from being fully understood, despite the increased literature in theoretical and empirical studies. For example, Evolution Strategies (ES) originated for numerical optimisation in real valued domain, and it remains unknown if some of the search strategies developed in that area apply to the discrete domain like COP [16]. The dynamics of many search strategies and parameter settings are still topics that attract interest from both researchers and industrial practitioners. Orthogonal Packing Problems are one of the basic models for testing and understanding complex algorithms, sharing the same combinatorial features as many other problems such as the Travelling Salesman Problems (TSP), Scheduling, and Timetabling, etc., but has its own

distinctive properties.

In this thesis, we addresses the use of Evolutionary Algorithms to solve Orthogonal Packing Problems. Our first aim is to contribute to the literature by using OPP as a test-bed to provide empirical studies that can help reveal important properties of EAs on the combinatorial search domain. We also design more efficient algorithms to provide high quality solutions that are suitable for use in the real world.

1.2 Contributions

The work of this thesis makes contribution to both the theory and its application, and in both meta-heuristics and orthogonal packing problems in general. From a theoretical perspective, several models and frameworks have been proposed to better understand the problem and the properties of different algorithms, and extensive empirical studies also provide relevant evidence:

1. We propose a framework to help understand the capability of multiple decoder hyper-heuristics in being able to search larger solution spaces than single-decoder meta-heuristics. The analysis of the framework is supported by strong evidence from empirical studies on a class of instances (see Chapter 3).

2. For the on-going debate on the Building Block Hypothesis of Genetic Algorithms, example cases are surprisingly hard to find [16]. We use OPP as a model to explicitly trace the Building Blocks (BBs) and find supportive evidence that BBs can effectively guide GAs into promising search areas. However, we also show the risk that the number of BBs may increase exponentially and eventually hamper the performance of GAs (see Chapter 4).

3. For orthogonal packing problems, especially multi-dimensional cases, phenotype representation is uncommon. The reason is due to the expensive computation involved in direct alteration of packings. We propose a phenotype representation that suits OPP of any dimension, as well as two generic operators, split and merge, that can manipulate the phenotype in efficient ways (see Chapter 5).

4. Evolution Strategies have been mainly applied to numerical optimisation in real-valued domains, or combinatorial problems which have simple representations. With the help of the phenotype representation and operators, our work is the first, as far as we are aware, that extends ES applications to the more complex domain of OPP (see Chapter 5).

5. The behaviour of ESs for combinatorial domains are much less understood than for real-valued domains. Discrete search space causes scalability issues and asymptotic success rate may not exist. Compared to real-valued domains, our study on ESs has revealed a different picture of the relationship between mutation strength and success rate for OPPs (see Chapter 5).

From an application perspective, we have designed algorithms that can provide high quality solutions for general OPP cases or address specific difficulties encountered for certain sub-classes of the problems.

1. The operators for adding and removing shapes from a packing, which we use throughout the thesis, is a unified approach that cope well with one to multi-dimension orthogonal packings.

2. The hyper-heuristic approach in chapter 3 can take advantage of multiple heuristics and solve more classes of OPPs than conventional meta-heuristics. The HH exhibits the same efficiency as standard approaches with the same convergence speed.

3. In chapter 4, the Grouping GA performs particularly well when high quality solutions are required and problem sizes are not too large. The best known results in the literature is able to solve benchmark instances to optimality for up to 20 shapes, while the GGA has pushed this boundary up to 40 shapes.

4. The self-adaptive ES and Grouping ES in chapter 5 are also efficient solvers and can be applied to a more general range of instances (even very large sized problems). They produce results comparable to some of the best results reported in the scientific literature.

1.3 Overview

Chapter 2 provides a survey of both the Orthogonal Packing Problems and the state-of-the-art algorithms for solving this problem. The chapter begins with a brief review of the definition and Mathematic Programming models for OPP, along with an introduction to the topology of the cutting and packing problems. The overview on models and topology makes clear both the similarity and distinction between OPP and other cutting and packing problems. In the following sections, we present the algorithms in four categories: exact algorithms, heuristics, meta-heuristics and hyper-heuristics. In particular, we discuss the strength and weakness of various current approaches, and also explain our motivation to set the stage for the algorithms we present in the rest of the chapters.

Chapters 3 and 4 consider approaches based on Genetic Algorithms. In Chapter 3 we develop a hyper-heuristic which utilises multiple low-level decoding heuristics. The hyper-heuristic is based on, and compared with, more conventional GAs. The difference is the hyper-heuristic approach utilizes multiple low-level heuristic decoders, which is developed specially to address a problem of many standard meta-heuristics that fail to search entire solution space due to the bias of using only one low-level heuristic. A theory framework of our hyper-heuristic approach

is discussed which explains the capability of the algorithm to search through a low-level heuristic search space. In the empirical study, we construct a set of new instances that standard GAs have difficulty to find the optimal while the hyper-heuristics achieves better results. The results for the new instances provide strong evidence that the hyper-heuristic can more effectively search the solution space. We also test benchmark instances from the scientific literature and show that the hyper-heuristic maintains the same efficiency as more standard approaches. Although the hyper-heuristic we developed is adopting GA as an example, the idea can be extended to benefit other meta-heuristics.

Chapter 4 is more focused on the search strategy of GAs, in particular, recombination of genetic encoding based on the Building Block Hypothesis (BBH) [87]. We investigate the Grouping Genetic Algorithms(GGA) which improves the performance of a standard GA by enhancing the genetic encoding with explicit encoding for BBs. The algorithm was first designed only for one-dimensional bin packing problems [66]. To extend it to suit more general cutting and packing situations, including higher dimensions and single bin (strip) packing, it requires a versatile definition of groups, therefore a new type of chromosome is introduced. The new genetic encoding varies in length and consists of individual shapes and blocks (groups of shapes which have no waste area). The blocks are explicit encoding for BBs, which the algorithm tries to discover during the evolutionary search process. The blocks, as partial solutions, also incorporate phenotype information into the genotype encoding. Compared to other algorithms, the GGA can successfully solve instances of much larger size. Our empirical study supports the claim that building blocks play an important role in GA evolution for OPP. However, we also show the number of BBs grows exponentially with increasing instance size, which decreases the performance of GGA if we do not contain the problem carefully. Potential improvements are suggested at the end of the chapter, one of which, Grouping Evolution Strategies, using more static grouping technique is

developed in the next chapter.

Chapter 5 is devoted to Evolution Strategies (ESs) which use phenotype representation as its search space, as opposed to genotype encoding in GAs. ESs are rarely used for cutting and packing problems, because ESs mutate phenotypes directly which is difficult for combinatorial optimization. In the literature, ESs theories are mostly derived from problems in continuous domains. For example, the asymptotic property of mutation strength underlying the standard search control strategy. To apply ESs to OPP, we first need to design novel, but still generic, neighbourhood search operators that suit the phenotype presentation. The central concern of this chapter is to find ES search strategies for combinatorial problems like OPP, which have very different characteristics than problems in continuous domains due to the highly interactive variables in solutions and in-differentiable stepping size in neighbourhood structure. By empirical studies, we discovered, surprisingly but not unreasonably, two clusters of promising settings for mutation strength in such a highly constrained and discrete search domain. Through further exploring the OPP fitness landscape, we derive a strategy called Grouping Evolution Strategy (GES), which groups shapes into static groups and applies different search strategy for different groups. Compared to other ESs, the GES outperforms other algorithms on most of the benchmark instances.

Finally in Chapter 6 we summarize our key findings and propose some suggestions for future research.

Chapter 2

Literature Review

2.1 Introduction

This chapter is presented in two parts; an overview of the problem and a survey of various algorithmic approaches. In the first part, section 2.2 describes the typology of Cutting and Packing Problems (CPP), and section 2.3 presents the modelling and formulation of these problems. The second part of the review consists a survey of exact algorithms (section 2.4), heuristics (section 2.5), meta-heuristics (section 2.6) and hyper-heuristics (section 2.7).

A typology provides a classification of various related problems. It is especially helpful to have a good typology as a roadmap to the vast amount of the papers in the literature on closely related topics. We review two typologies proposed by Dyckhoff [60] and Wäscher et al. [178], which provide the context of our research and make clear the relationship between other classic problems and the Orthogonal Packing Problems (OPP) that will be studied in this thesis. Some of the OPP (especially one-dimensional problems) can be expressed in a mathematical programming (MP) formulation. The mathematical programming formulation captures essential aspects of these problems and can be tackled by efficient algorithms when instance size is limited. When dealing with higher dimensional

problems, it is much harder to formulate and solve the problems with mathematical programming, due to the increase in the number of variables and constraints. Consequently, researchers often resort to alternative models to understand the properties of the problems. A useful model is proposed by Fekete and Schepers [74] which utilises interval graphs as an abstraction of a class of equivalent packings.

Many of the exact algorithms are indeed based on the MP formulations presented in the previous section. The purpose of the review is to highlight the advantages and disadvantages of the exact methods. Especially, unless a problem has special structures that submit itself to linear (sub-linear) algorithms, most of the OPPs belong to NP; and unless N = NP, most of the OPPs can only be solved to optimality when the problem sizes are relatively small.

In section 2.5 we review heuristic approaches to OPPs. Heuristic approaches are studied by researchers for two purposes. First, as an alternative to exact methods, they provide fast solutions to instances even with very large sizes. Examples include greedy strategies such as First Fit(FF) and Best Fit(BF) [40]. Some heuristics for certain classes of problems have been proved to be bounded with approximate ratios, constant factors to optimal [40]. On the other hand heuristics normally generate solutions with good quality, even if they cannot guarantee optimality or bounds. For this reason, heuristics are often utilised by meta-heuristic approaches as a sub-routine which we will discuss in section 2.6.

Meta-heuristics iteratively search for, and hopefully improve, existing solutions based on certain search strategies. We concentrate on two types of strategies; Genetic Algorithms (GA) and Evolution Strategies (ES), while also reviewing many other different strategies. Most of these seemingly different strategies are actually based on a similar idea that a successful search strategy needs to balance the process of exploration and exploitation. Indeed, De Jong provides a unified view of various approaches [54], and another type of more general approach, hyper-

heuristics, have emerged in recent years that attempts to hybridise and raise the generality of search methodologies [23].

GA and ES belong to the relatively new paradigm of search strategy known as Evolutionary Algorithms (EA). One of the distinctive feature of these algorithms is that they maintain a population of candidate solutions and draw upon historical and parallel information to carry out the search process. We will review the theories of these algorithms, which also includes some studies not in OPP domain but illustrates the behaviour of EAs more clearly. We also examine previous applications of the EA strategies being applied to OPP.

2.2 Typologies

According to Haessler and Sweeney [98] the earliest formulation of stock cutting problem can be traced back to 1939 by the Russian economist Kantorovich [119]. Since then the problem has evolved into a large family of problems with quite diversified objective functions and constraints. On the application side they cover many real-world problems and often appear under different names, such as the knapsack problem [161], stock cutting [80], trim loss [61, 105], bin packing [39], container loading [160], and multi-processor scheduling [78], etc. (formal definitions of many of these problems will be introduced in section 2.3). Most of these problems remain open due to them being NP-hard [79] even for the one dimensional problem.

A difficulty we face in structuring a literature review for the cutting and packing problems is the various ways we could categories the problems. Almost all articles can be looked at from different angles and can be categorized in at least one of the following ways:

- according to problems tackled, e.g. is it about one-dimensional knapsack problem, two-dimensional stock cutting, or any other variant problems;

- according to topics addressed, e.g. is it introducing new modelling methods, algorithms, or analysis of performance of existing techniques; according to techniques applied, e.g. is it proposing new exact or heuristic or iterative searching methods.

In fact, classifying the problems is itself a complex task, and two papers [60, 178] are dedicated to this typology specifically. In 1990 Dyckhoff published the first proposal [60]. He has observed many important characteristics of Cutting and Packing Problems: dimensionality, quantity measurement, shape of objects, assortment, availability, pattern restrictions, assignment restrictions, objectives, states of information and variability. Despite the complexity, he tries to simplify the categories by concentrating only on 96 types formed by combinations of four main characteristics:

Dimensionality :

(1) One-dimensional.

(2) Two-dimensional.

(3) Three-dimensional.

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

Kind of assignment :

(B) All objects and a selection of items.

(V) A selection of objects and all items.

Assortment of large objects :

(O) One object.

(I) Identical objects.

(D) Different objects.

Assortment of small items :

(F) Few items (of different figures).

(M) Many items of many different figures.

(R) Many items of relatively few different (non-congruent) figures.

(C) Congruent figures.

The above system is easy to use. However, it also causes ambiguity as some problems cannot be put into a unique category. More recently Wäscher et al.[178] took a more refined approach, which firstly excludes problems having other additional aspects than pure cutting and packing problems. For pure cutting and packing problems, the following characteristics are considered for classification:

- *kind of assignment* whether the objective function is a maximisation function (when only a subset of small items can be accommodated, e.g. knapsack problem), a minimisation one (when all small items need to be packed with least cost) or multi-objective optimisation;
- *assortment of small objects* whether small items are identical in sizes, weakly heterogeneous or strongly heterogeneous;
- *assortment of large objects* whether the container(s) is(are) rectangular, homogeneous or inhomogeneous. This characteristic separates problems into different categories, e.g. pallet loading (one large object with fixed sizes), strip packing (one large object with infinite sizes on some dimensions), bin packing (multiple large objects with identical sizes);
- *dimensionality* whether the problem is of one, two or more dimensions;
- *shape of small items* whether the small items is orthogonal or non-orthogonal. This characteristic distinguishes cases between regular and non-regular shapes.

For refined problems, if there are no additional constraints, is it then one of the 'standard' types; otherwise further steps are taken to put it in one of the special

problems. The scientific papers from 1995 to 2004 have been classified according to this system. New papers, along with a categorized bibliography can be found on the ESICUP website (`http://paginas.fe.up.pt/~esicup/`).

Another classification method has been presented by Sweeney and Paternoster[174]. They collected more than 400 papers, omitting those on complexity and worst-case analysis, and assigned them into categories forming a 3x3 matrix. Rows of the matrix are defined by dimensionality from 1 to 3, columns are formed by solution approaches: sequential heuristics, single pattern oriented (e.g. dynamic programming based) algorithms, and multi-pattern oriented (e.g. math programming based) algorithms. Some of the mostly studied topics are listed separately, such as assortment problems, and multi-stage problems. A follow-up work [62], in 1997, selected more than 150 annotated references on various topics grouped into several topics, which are based on the dimensionality of the problems.

In 1979 Hinxman [105] published a survey on trim-loss and assortment problems, considering problems of 1-dimension, 1.5-dimensions and 2-dimensions. Haessler and Sweeney published updated reviews [97, 98] in 1991 and 1992. They gave detailed mathematical programming formulations of these problems and their variations. In particular, they explained the differences in staged, guillotine/non-guillotine, non-orthogonal pattern generation.

Dowsland and Dowsland [56] reviewed various classes of 2-dimensional problems, including guillotine and non-guillotine patterns, bounded and unbounded pieces of small items, pallet loading, bin packing and strip packing. They also briefly discussed works up to 1990 on 3-dimensional and non-rectangular problems.

Other surveys and reviews on specific sub-area are: [40] on worst-case and average-case analysis of packing heuristics for one dimension problems, [145] on empirical study of exact algorithms of 0-1 variant knapsack problems, [76] on multidimensional 0-1 knapsack problems, [135, 136, 139, 145] on two dimensional

problems (including [107] on genetic algorithms), [154] on three dimensional problems and [15] on irregular shape nesting problems.

2.3 Orthogonal Packing Formulation

2.3.1 One-dimensional Problems

The mathematical formulation of the one-dimensional problems is usually given as a linear program.

1. The classical 0-1 Knapsack Problem (KP) is defined in [147] as a set of items $i \in (1, 2, \cdots, n)$ each of which has associated values of cost c_i and profit p_i, the objective is to choose a subset of items to maximize the total profit while keeping the cost under a capacity limit, C, of the knapsack, i.e.

$$max \quad \sum p_i x_i \tag{2.1a}$$

$$s.t. \quad \sum c_i x_i \leq C \tag{2.1b}$$

$$x_i \in (0, 1) \tag{2.1c}$$

   for $i = 1, 2, \cdots, n$. x_i is a binary variable indicating if item i is selected into the knapsack. The inequality constraint 2.1b specifies total cost of selected items must be equal or less than C.

2. Bounded or constrained Knapsack Problems [159]: This is the first variant of classical KP, by simply changing equation 2.1b to

$$x_i \in \{a, b\} \tag{2.2}$$

   where a, b are non-negative integers and $a \leq b$;

3. Unbounded Knapsack Problems [149]: This is another variant of KP, by

changing equation 2.2 to $a = 0, b = \infty$.

4. Multiple Knapsack Problems (MKP) [149]: This is also called the 0-1 multi-knapsack problem (or multi-dimensional 0-1 KP in some of the literature, we do not use multi-dimensional 0-1 KP to avoid confusion with geometrical multi-dimensional problems). It is also one of the most extensively studied problems. Instead of having a single capacity we have multiple bins, therefore multiple capacities. Instead of one limit of C, we have a set of limits C_j ($j \in \{1, 2, \cdots, m\}$) for each knapsack, accordingly equation 2.1b is then changed to

$$\sum c_{ji} x_i \leq C_j \tag{2.3}$$

and normally we assume $c_{1i} = c_{2i} = \cdots = c_{ji}$ (which means unit cost of an item does not change whichever knapsack it is put in).

5. Stock Cutting Problems (SCP) [176]: This type of problem is slightly different from the knapsack problem. The standard one-dimensional version can be described as, a stock of rods with the same length C have to be cut into m different smaller pieces of length c_i $i \in \{1, 2, \cdots, m\}$, and least q_i piece of item i is required. The objective is to minimise the total number of stocks used.

   In the seminal paper of Gilmore and Gomory [81], they proposed a innovative formulation and a column generation method to tackle the problem. They used column vectors $[a_{1j}, a_{2j}, \cdots, a_{mj}]^T$ to represent j different patterns, with $a_{ij}, i \in \{1, 2, \cdots, m\}$ denotes the number of item type i in the jth pattern. Integer variable x_j indicates the number of pattern j in the solution. The formulation is written as

$$min \quad \sum_j x_j \tag{2.4a}$$

$$s.t. \quad \sum_j a_{ij} x_j \geq n_i \tag{2.4b}$$

$$x_j \geq 0 \tag{2.4c}$$

6. Trim Loss: This problem relates to the SCP by altering the objective function to minimizing the total wasted "trims".

7. Bin Packing Problems (BPP) [40]: BPP are also closely related to SCP. The bins (analogues to stocks) often have the same capacity of C and the objective is to minimize the total number of bins needed.

$$min \quad \sum_{j=1}^{n} y_j \tag{2.5a}$$

$$s.t. \quad \sum_{i=1}^{m} c_i x_{ij} \leq C y_j \tag{2.5b}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} = 1 \tag{2.5c}$$

$$x_{ij} \in \{0, 1\} \tag{2.5d}$$

$$y_j \in \{0, 1\} \tag{2.5e}$$

for $i \in \{1, 2, \cdots, n\}$, and $j \in \{1, 2, \cdots, m\}$, n and m are the number of items and bins respectively. The binary variables x_{ij}, y_j are defined in the way that if bin j is used in packing $y_j = 1$, otherwise $y_j = 0$; similarly if item i is placed in bin j then $x_{ij} = 1$, otherwise $x_{ij} = 0$. Therefore, the meaning of 2.5c is that each item must be placed in exactly one of the bins, and in 2.5b all items in bin j must not exceed the total limit of that bin.

The above are all strict integer programming models commonly found in the literature. Other innovative approaches of mathematical programming have also been studied, such as in [59]. However, the mathematical formulations are difficult to solve in most cases due to the size of the search space. Nonetheless early research still found ways of applying mathematical programming, such as column-generation[81, 82, 84], to resolve small-sized problem instances as we will discuss in the algorithm section (section 2.4). In [31] Caprara provides a detailed discussion on the properties of integer programming (IP) and associated relaxations. It is also popular to use relaxed integer programming formulations to obtain lower/upper bounds.

Beside mathematical programming models, heuristics can generally be regarded as another way of modelling problems (as well as solution approaches) with a view to obtain approximate results within reasonable computational times. These will be introduced below in the heuristic algorithms section (section 2.5).

2.3.2 Multi-dimensional Problems

When extending cutting and packing problems to two or more dimensions, some new difficult issues arise. The first issue is the exponentially increasing number of variables and constraints associated with the higher dimension, which is basically due to these being NP problems. It is also more difficult to generate feasible patterns. This means the feasible search space is harder to define, because to maintain a feasibility in one dimension, we are normally subject to restrictions in other dimensions. And such restrictions change dynamically as packing procedure progresses. There are also many interesting variations of the problem, particularly those requiring certain patterns of layout to be generated.

In a basic two-dimensional problem both small shapes and large containers have associated widths and heights. The following are some of the most popular problem variants that have been studied:

Bin Packing, Strip Packing These categories are extensions to the one-dimensional problems under the same names, and can be modelled by classical LP methods and by introducing variables on each dimension (see below). Thus a two-dimensional bin packing problem refers to a minimization problem in the context of two dimensions, while the multi-dimensional stock cutting is to maximize profit value. Multi-dimensional strip packing can be viewed as a special case of multi-dimensional bin packing, where only one large container with infinite height is provided, and the objective is to minimize the total height of the layout.

Orientation (Rotation) If the width and height of a small item can be exchanged, we say the problem allows rotation or that the items are non-oriented. Otherwise we say the problem is of fixed orientation.

Guillotine and non-Guillotine Guillotine pattern can be found in many real-world cutting applications, such as paper and glass industries. It requires a cut goes from one edge to an opposite edge. Non-guillotine patterns are free from such a restriction.

Tetris As suggested by its name (after the famous game), some applications require items start from top and, before settling permanently, move in any direction except trespass on space already occupied by other items. Such cases can be easily found in many industrial applications, such as shipping container loading.

Pallet Loading Sometimes items are of homogeneous nature and can be rotated. The objective is to load as many of these small items as possible.

Due to the strong connections between one-dimensional and multi-dimensional problems, classical linear programming and integer programming models can be extended to higher dimensional problems. In fact, the first integer programming

model for one-dimensional problem (in [81]) was soon applied to two-dimensional problems by the same authors in [83]. To limit the complexity of defining patterns, the authors studied a special two stage guillotine cut.

Another approach taken by Beasley [12] and Hadjiconstantinou et al. [95] can be more generally applied to non-guillotine patterns. The model uses Euclidean coordinates to explicitly define layout patterns. Let binary variable x_{ipq} be equal to 1 if item i is placed with its bottom-left corner at coordinate of (p, q), and 0 if otherwise. The 0-1 integer programming can be written as follows, where v_i and a_{ipq} are associated profit and cost of item i:

$$max \quad \sum_{i \in M} \sum_{p \in L} \sum_{q \in W} v_i x_{ipq} \tag{2.6a}$$

$$s.t. \quad \sum_{i \in M} \sum_{p \in L} \sum_{q \in W} a_{ipq} x_{ipq} \leq 1 \tag{2.6b}$$

$$P_i \leq \sum_{p \in L} \sum_{q \in W} x_{ipq} \leq Q_i \tag{2.6c}$$

$$x_{ipq} \in \{0, 1\} \tag{2.6d}$$

for $i \in \{1, 2, \cdots, m\}$ (or the set M), $p \in \{1, 2, \cdots, l\}$ (or the set L) and $q \in \{1, 2, \cdots, w\}$ (or the set W). P_i and Q_i are lower and upper bounds of the quantity of item i.

The above LP models suffer from a large number of variables and the problem of redundancy. To avoid these problems, another novel approach by Fekete and Schepers [71–74], proposed in 1997, takes advantage of interval graph theory to from a class of packing patterns. These patterns, though having different internal layout, form enclosed rectangles with equal widths and heights. The idea is for each pattern, we can always project it to a x and y axis, with item width and height as intervals along the two axis respectively. The following example (see Figure 2.1) from their papers illustrates 5 items v_1, v_2, v_3, v_4, v_5 packed as the pattern shown, and their projection on x and y axes can be presented in a pair of interval graphs,

one for each dimension. The nodes in an interval graphs represent items, and an edge exists between two nodes if their projection on the dimension overlap.

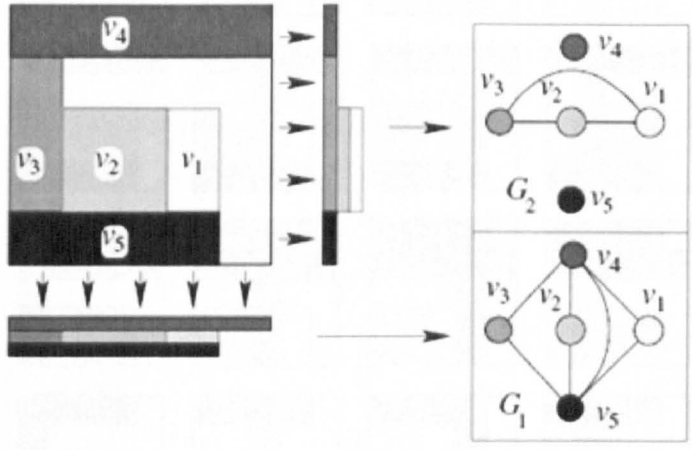


Figure 2.1: A two-dimensional packing and the interval graphs G_1, G_2 induced by the axis-parallel projections (from [72])

The formal expression of graph presentation is: Let $G_i = (V, E_i)$ be induced graph from projection i ($i = 1, 2, \cdots, d$), (for two-dimensional problems $d = 2$), the set of graphs for d dimensions form a feasible packing class if and only if they satisfy the following properties: P1: Each $G_i = (V, E_i)$ is an interval graph; P2: Each stable set S of G_i is x_i-feasible, i.e. where $p(b)$ is the starting point of item b and $w(b)$ is the length of b; W is the total length of bin. P3: $\cap_{i=1}^{d} E_i = \emptyset$ An advantage of this presentation is a pair of interval graphs can actually represent more than one packing, but a whole class of packing, which are listed below (figure 2.2). This is a desirable thing as it can greatly reduce duplication of equivalent patterns.

This model has two major drawbacks. First, it may be the case that search over the graphs rather than binary variables, can effectively reduce redundancies. However, unfortunately, the basic NP-hard problem still exists as it relies on checking if each graph forms an interval graph, and this decision problem is still open in graph theory. Secondly, treating the shapes indifferently as nodes (non-weighted nodes) is ignoring the size information of each shape. While this may

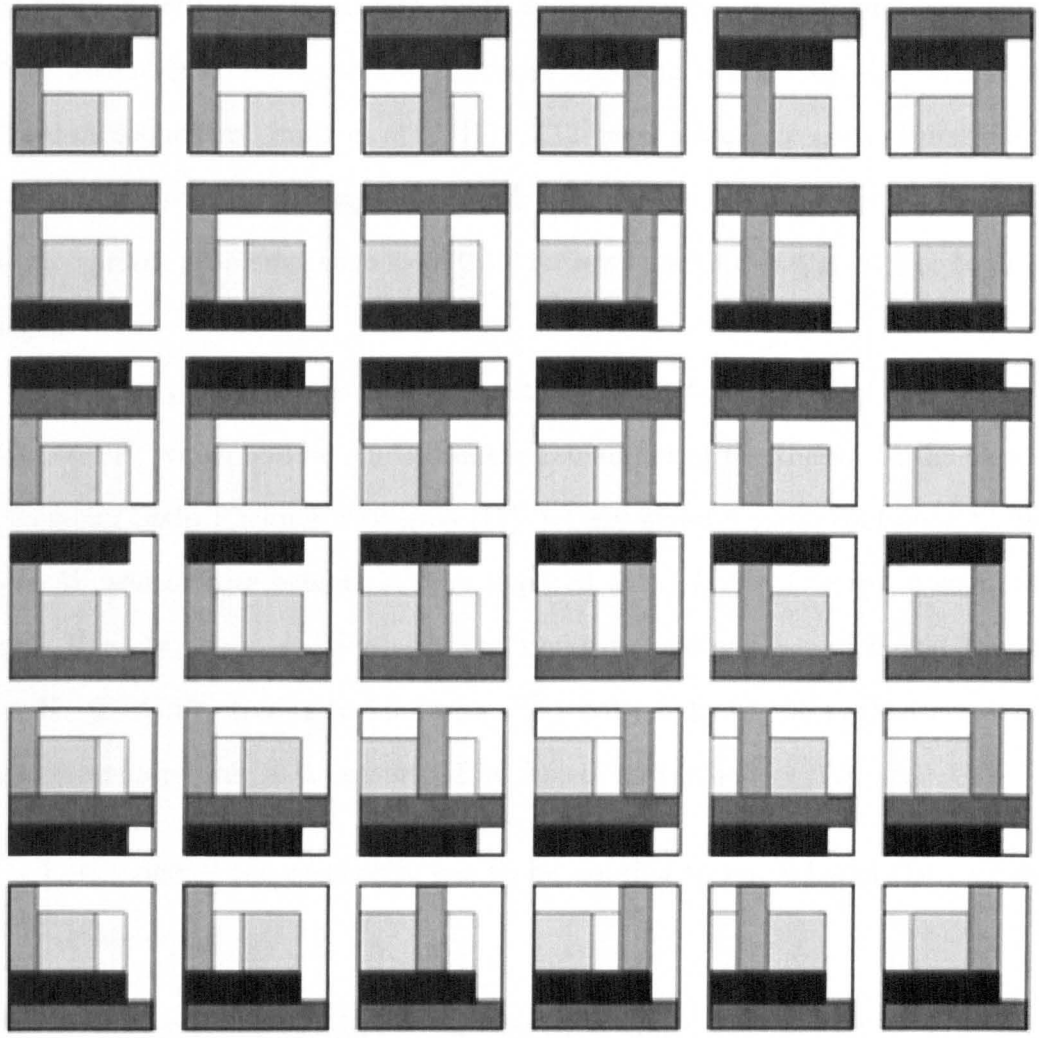


Figure 2.2: All feasible packings of a packing class have the same interval graph representation as in 2.1 (from [72])

simplify the model, it can cause infeasible solutions in degenerated cases details will be discussed in chapter 5 (Figure 5.1). Therefore, we believe this is a flaw in the original publication when applying this approach to construct a packing from a pair of interval graphs. However, the interval graphs are still a good abstraction of packing class, which can be useful to some extent. For example, we can use the graphs as an abstraction to compare similarity between two packings (see chapter 5 for details.)

There are some other modelling methods exist in the literature which are less studied, such as a block structure proposed by Mukhacheva and Mukhacheva [153].

The block structure uses virtual cuts to split packings vertically and horizontally before assigning to each piece a notion of ordering defined by the vertical and horizontal sequences. Imahori et al. [111, 112] used a sequence pair representation which is the orders of a shape on x and y dimension. There are also models for specific variant problems, such as in [140] for level packing and in [37] for container loading.

In the last part of this section, we define the Orthogonal Strip Packing Problems (OSPP) which will be studied in the remainder of the thesis. In the classical Orthogonal Strip Packing Problems [11], we are given a large container $\mathbf{C}$, with width $\mathbf{W}$ and infinite height. We are required to pack into $\mathbf{C}$ a set of small rectangles $\mathbf{R} = \{r_1, r_2, ..., r_n\}$ with (w_i, h_i) denoting the width and height for each $r_i \in \mathbf{R}$. The objective is to minimise the total height of the packed rectangles. Typical assumptions, as summarized by Fekete and Schepers [72], are:

1. Each edge of the rectangles has to be parallel to one edge of the container (orthogonal);
2. We do not require guillotine cutting (free-from);
3. All rectangles must be within the container (closeness);
4. Rectangles must not overlap with each other (disjoint);
5. Rectangles cannot be rotated (fixed orientation).

The problem can be classified as two-dimensional regular open dimensional packing (2D-R-ODP) according to the typology proposed by Wäscher et al.[178]. As with its one dimensional counterpart, the two-dimensional orthogonal packing is also strongly NP-hard [11].

2.4 Exact Algorithms

Many exact algorithms have been used to tackle one-dimensional problems. Most of these algorithms are based on the integer programming models in the previous section. There are two main issues for these models. The first is how to deal with large instances. The second is how to search through the feasible solution space effectively. Here we review the most important methods to tackle the difficulties involved in integer programming: column generation and branch and bound. We will also look at methods for computing bounds separately.

2.4.1 Approaches to One-dimensional Problems

Column generation To address the large-scale issue, in early 1960s Gilmore and Gomory [81] first presented a column generation method to trim loss problem. They used a column vector to represent a possible cutting pattern from one stock rod, which has m elements each represents how many pieces of the mth type is cut from the pattern. The problem is then decomposed to two stages, the first is to decide if a pattern is feasible, the second is to decide if a pattern should be added into the solution so as to minimise the waste. To choose a pattern for a subsequent LP, they first solve an auxiliary knapsack problem which can be solved more easily. Haessler in [96] improved the methods by placing restrictions on the coefficients, which led to solutions with fewer patterns and was easier to round to integer values.

Branch and bound Branch and bound algorithms are probably one of the most widely used exact algorithm. The basic method can be illustrated as a tree search. The tree is constructed from the root, which in an initial state with some variables being arbitrarily fixed to certain values. Along branches the search space is partitioned into disjoint parts, as each branch explicitly includes or excludes certain conditions. The search starts from the root,

and prunes the tree, testing on upper or lower bounds (depending on the objective function) to avoid enumerative evaluation on each leaf node. It has been noticed by many the performance of this type of algorithm depends heavily on how tight the bounds are. In addition, calculating the bounds is often computationally expensive.

Bounds

To derive an effective bound, Martello and Toth [147, 148, 151] used a type of continuous upper bound for the knapsack problems. The bound is obtained by sorting items then placing them successively into the knapsack according to the ratio of profit p to cost c of all n items:

$$\frac{p_j}{c_j} \geq \frac{p_{j+1}}{c_{j+1}} \tag{2.7}$$

(for each item $j = 1, \cdots, n-1$) and until no more items can be placed, a critical point s is defined as:

$$s := min\{i : \sum_{j=1}^{i} c_j > C\} \tag{2.8}$$

where C the total capacity of the knapsack.

If there is any unutilized space, it can be filled by an item with the biggest unit value of $\frac{p_s}{c_s}$, (since more valuable items before the critical point are all used up). The bound can be written as:

$$\left\lfloor \sum_{j=1}^{s-1} p_j + (C - \sum_{j=1}^{s-1} c_j)\frac{p_s}{c_s} \right\rfloor \tag{2.9}$$

Lower bounds for the bin packing problems (as we are dealing with minimization problem) can be computed in a similar fashion, though other methods exist in the literature, such as Lagrangian, and surrogate relaxation of equation 2.5 (in section 2.3). The first trivial bound is the sum of total volume of small items,

according to [150], which has computational complexity of $O(n)$ and worst-case performance of $1/2$. And this bound $L_1()$ dominates the Lagrangian relaxation.

$$L_1 = \left\lceil \sum_{i=1}^{n} c_i / C \right\rceil \tag{2.10}$$

Another tighter bound is obtained through a very similar idea for the knapsack problem. It takes advantage of items larger than half of the bin size, such that no two items can be placed together. The remaining space of used bins is filled by smaller items. If any mid-sized items remain unpacked, they open up new bins with lower bound given by (2.10). Mathematical expression of the idea is, for item set I any arbitrary value $\varepsilon \in [0, 1/2]$:

$$L_2(I) = \arg \max_{\varepsilon \in [0,1/2]} L_2(I, \varepsilon) \tag{2.11a}$$

$$L_2(I, \varepsilon) = |\{i \in I | c_i > 1 - \varepsilon\}| + L_1(\{i \in I | \varepsilon \leq c_i \leq 1 - \varepsilon\}) \tag{2.11b}$$

(where I is the set of all items and $|\{i \in I | c_i > 1 - \varepsilon\}|$ means the number of items whose volume exceed the threshold $1 - \varepsilon$). According to [150] the worst-case performance is $2/3$ and the complexity is $O(n)$. $L_2()$is improved by Labbe et al. in [129] by further exploiting the characteristics of items within the range of $1/3$ to $1/2$ of bin size, as two of such items can be put together into a bin. Define a set I_3 for these items, the expression of the bound is:

$$L_3(I) = \arg \max_{\varepsilon \in [0,1/2]} L_3(I, \varepsilon) \tag{2.12a}$$

$$L_3(I, \varepsilon) = |\{i \in I | c_i > 1 - \varepsilon\}| + \left\lceil \frac{I_3}{2} \right\rceil + p(\varepsilon) \tag{2.12b}$$

$$\textit{where} \quad p(\varepsilon) = \max \left(0, \left\lceil \sum_{c_i \in [\varepsilon, 1-\varepsilon]} c_i - |\{i \in I | c_i \in \left[\frac{1}{2}, \varepsilon\right]\}| - \left\lceil \frac{|I_3|}{2} \right\rceil \right\rceil \right) \tag{2.12c}$$

In [20] Bourjolly and Rebetez proved the complexity of this bound to be

$O(nlogn)$ when items unsorted and $O(n)$ when sorted non-increasingly; and worst-case performance is 3/4.

Another interesting fast bound introduced by Fekete and Schepers in [70] which made use of the idea of Dual Feasibility Function (DFF) by Johnson in his PhD thesis [115]. DFF is a type of mathematical transformation function which maps an original hard problem into an easier intermediate problem. The function applied in [70] is defined as a function $u : [0, 1] \rightarrow [0, 1]$ if for a finite set of F the following two inequalities hold:

$$\sum_{x \in F} x \leq 1 \tag{2.13a}$$

$$\sum_{x \in F} u(x) \leq 1 \tag{2.13b}$$

The bound can be computed in $O(pn)$ time.The DFF approach for deriving bounds has also been taken for higher dimensional problems, and further developed by Carlier et al. [33].

Other methods of computing bounds can be found in two recent surveys [46] and [45].

Other Exact Methods for One-dimensional Problems

There are some other approaches, mainly hybrid methods, that are worthy of mention. In [14] Belov and Scheithauer show a way to combine the branch and cut and price approaches which leads to an enhanced search process. Dynamic Programming (DP) is generally an enumerative procedure and therefore not normally used for problem sizes larger than 20. However, in [144, 158] Pisinger et al. demonstrated one way of using DP to start the search from a core area. The search first starts from a so called Dantzig integer solution given by formula 2.7 and 2.8, and at the beginning point the incumbent solution only includes one item s. The DP is constructed by trying to add $s - 1$ and $s + 1$, then $s - 2$ and $s + 2$

and so on. The complexity is $O(cn)$ (where c is a constant and n is the size of a problem) which enables the method to provide reasonable solutions for instances of 10,000 items within a few minutes.

2.4.2 Approaches to Multi-dimensional Problems

As with 1D problems, exact algorithms for 2D problems mainly involve branch and bound and other enumerative techniques, e.g. dynamic programming.

In [146] a search tree is formed by placing or removing an item on the so-called corner points on an 'envelope'. Some procedures are taken to prevent from producing duplicate patterns.

Similar approaches have been taken in [143, 152] where items are sorted first according to their height or area. Searching is carried out in a two-stage fashion: with a first level branch to decide if an item is in a certain bin or not; followed by a second level branch to decide the exact position of the item within the bin. The second level can be replaced by heuristic methods to accelerate the procedure by sacrificing a certain degree of accuracy.

To follow the interval graph representation, Fekete and Schepers in [69, 74] built a binary tree by including and excluding edges from each node. The search outperforms previous methods by quite a large margin.

As with 1D problem, deriving proper bounds is crucial to many algorithms, especially when involving branch and bound techniques. Many of the bounds regarding higher dimensions are extensions of the principles of 1D problems. Further readings can be found in [70, 140, 143].

2.5 Heuristics

2.5.1 Application on One-dimensional Problems

Rounding LP Relaxation The idea for this is straightforward. It takes two phases, and in the first it finds optimal value of a Linear Programming relaxation which is easier to solve than the Integer Programming formulation. In the second phase it searches the neighbourhood of the relaxed optimal solution. In [169, 170] Scheithauer and Terno presented a theoretical investigation on the Modified Integer Round-up Property (MIRUP) for one-dimensional stock cutting problems. In [13] Belov modified the rounding heuristic and combined it with Chvatal-Gomory cutting planes and column generation to tackle multiple stock lengths in the one-dimensional cutting stock problem.

Next Fit (NF) This heuristic was first described in [115], and is a so-called online bounded space algorithm. Each piece arrives one after another (i.e. the algorithm is not aware of the preceding pieces). When one piece arrives it has to be placed into the current open bin. At any given time only one bin is available for packing. If the current piece does not exceed the capacity of the current bin, the piece is placed in it and the algorithm proceeds to next incoming item; otherwise the current bin is closed and a new bin is opened. According to Johnson [115], the worst case ratio is $R_{NF}^{\infty} = 2$. (The notion means when item number tends to infinity, the NF algorithm has its worst to optimal ratio approaches to a limit of 2. The similar notion applies to the other algorithms listed below.)

First Fit (FF) FF is also an online algorithm, but unlike NF, it is an unbounded space algorithm, i.e. all partially filled bins are available for packing. Actually it indexes all bins according to the sequence they employed and puts

the current piece into the lowest indexed bin which is large enough. FF runs in $O(n \log n)$ time, and the worst case ration is $R_{FF}^{\infty} = \frac{17}{10}$ [78] .

Best Fit (BF) This packing heuristic checks all partially filled bins which are large enough for the current piece. The piece is assigned to the bin having the smallest residual space. It can be implemented in $O(n \log n)$ time. For a specific instance it may behave differently from FF [117], but the FF worst case is also hold for BF, i.e. $R_{BF}^{\infty} = R_{FF}^{\infty} = \frac{17}{10}$.

Other Simple Online Algorithms Several other online algorithms have also been studied, mainly for theoretical analysis of complexity and performance ratio. So we simply list their definitions here, Worst Fit (WF) fits next item into the partially filled bin with the lowest level. Almost Worst Fit (AWF) fits the next item into the partially filled bin with second lowest level. Any Fit (AF) is a generalized algorithm, which does not start a new bin unless no current partially filled bins can be used. Almost Any Fit (AAF) never packs items into the lowest partially filled bin unless there is more than one such bin. Performance ratios can be found in [116, 117].

Bounded Space Algorithms Bounded space means only some of the partially filled bins, instead of all of them, are available for packing. These are variant versions of NF, FF and BF, respectively. A constant K represents the number of bins can be kept open at a time. The following relationships hold [48, 50, 142],

$R_{NFK}^{\infty} = \frac{17}{10} + \frac{3}{10(K-1)}$

$R_{FFK}^{\infty} = \frac{17}{10} + \frac{3}{10K}$

$R_{BFK}^{\infty} = \frac{17}{10}$

Harmonic Algorithms These are also a set of bounded space algorithms first introduced by Lee and Lee [131]. The principle is to divide items and bins

into K types according to their sizes, say $I_k = (\frac{1}{k+1}, \frac{1}{k}]$. Bins with type k only receive items of the same type. Woeginger et al [1] uses another improved version called Simplified Harmonic K (SHk), which uses a more complicated type of structure. As with Lee and Lee, van Vliet at el. presented the lower bounds for different K in [1, 36, 131, 177].

According to a theorem in [131], online bounded-space algorithms have $R_A^\infty \geq 1.69103$. In fact the ratio can be approximated at the limit. Until in [77] when a repack is allowed, and in an unpublished script by Grove where look-ahead is allowed, the ratio can be guaranteed.

Other Arbitrary Online Algorithms There are some other online algorithms which can improve those relatively simple ones introduced above: Group-X Fit (GXF) [116], Refined First Fit (RFF) [182], Refined Harmonic (RHk) [131], Modified Harmonic (MHk) [163]. A good summary of them can be found in [40].

Offline Algorithms These algorithms are allowed to chose any item freely and pack it to any available bins that are large enough. Among those the mostly notable ones are First Fit Decreasing (FFD) and Best Fit Decreasing (BFD). They both sort items by a non-increasing order and pack them according to rule of FF and BF correspondingly. It has been approved in [9, 116] $R_{BFD}^\infty = R_{FFD}^\infty = \frac{11}{9}$.

Sum-of-Squares (SS) The algorithm is firstly presented by Csirik et al. in 1999 [52]. It performs remarkably well in terms of performance ratio and complexity though the idea is quite simple to explain. SS works as an online algorithm and displays some good self organizing properties. The fitness function it uses to decide where to put an item is based on $\sum_{k=1}^{B-1} N_p(h)^2$, where B is the bin number, and $N_p(h)$ is the number of bins with height h for the current partial packing p. According to [48, 51], it runs in time of

$O(nB)$. For any discrete distribution in which the optimal expected waste is sub-linear, SS also has sub-linear expected waste.

2.5.2 Application on Multi-dimensional Problems

Some heuristics for one-dimensional cases can be modified for the strip packing problem. Baker et al. [11] presented a bottom-up left-justified (BL) heuristic, which finds the lowest feasible space, similar to one dimensional First Fit (FF), and packs left justified. Two other heuristics have been described by Liu and Teng [134]. Rectangles are dropped from the top right corner of the container, as in a Tetris game, and moved downwards then leftwards until it settles at a stable position. Contrary to BL, these two heuristics are both top-down left-justified, which overlook any holes formed by preceding rectangles in the partial packing. Therefore we regard them as analogues to Next Fit (NF) which never utilise empty spaces produced at an earlier stage. No heuristics have strictly adopted the Best Fit (BF) policy, which fits a piece into the smallest feasible space. Hayek et al. [63] proposed a heuristic related to BF, which matches pieces to available space based on an assessment of the fitness of both width and height. A few offline heuristics have also been extended for higher dimensions, notably Gu et al.[93] proposed a Next Fit Decreasing Height (NFDH), Burke et al. [28] designed a Best Fit Decreasing Width (BFDW) heuristic. Some heuristics have been specially designed for certain cases, such as packing pieces onto shelves [7, 8, 32, 53]. Other classes of one-dimensional heuristics are yet to be developed for higher dimensional problems, such as Harmonized Fit [179] which classifies pieces and available spaces into several types, (e.g. big, middle and small) and packs pieces into spaces of the same type. It is worth noting that a lot of analysis in relation to worst-case and average-case performances has been carried out on one-dimensional cases. One important conclusion is that no heuristic consistently outperforms any other for all classes of problems. Examples are given in [40],

for both online and offline heuristics, worst-case and average-case restricted to uniform distribution of items' size. For higher dimensional problems, although they are less thoroughly understood and fewer results have been reported so far [10, 32, 53, 93, 121, 173], similar observations have been noted, e.g. for some ordered lists Best Fit generates better packings than First Fit, while for some other lists the opposite is true.

Compared with 1D problem, some new concepts appear in higher dimensions. They are mainly for the purpose of pattern generation which is not an issue in 1D.

Bottom Left (BL) Many of the 2D heuristics assume items are bottom-left (BL) justified when finding a placement point, i.e. items are put into a bin one by one, at the lowest and left-most position according to the states of partial packing when each piece arrives (see [11]). It is easy to see the BL is an online algorithm. The worst case performance is 3 for the BL algorithm. In Chazelle's [35] implementation the algorithm runs in $O(n^2)$.

Bottom Left Fill (BLF) By looking at the patterns generated by the BL algorithm, one may have an instant impression that this algorithm creates a lot of empty space, which can be better utilized by placing small items into these holes. The idea results in an improved heuristic [4] which may have a better performance ratio. We have not found any report on proving worst or average case ratio for BLF. But our empirical study shows BLF performs better than BL, which also agrees with our intuition explained above.

Best Fit (BF) The algorithm [28] follows the above thinking, i.e. minimize wasted space at each step of packing (creating as few holes as possible). It does this by looking for information on current partial packing and all items left unpacked, and finds for each piece the best matching space. To reduce running time, items are sorted by non-increasing height. By an empirical study we found BF is usually better than BL and BLF, though run times

inevitably increase (as it is an off-line algorithm requires more information to process from each step of packing).

Lowest Fit Left Right Balanced (LFB) In [183] a heuristic is proposed that finds the lowest feasible position for a shape and align it to left or right so that the distance to the edge is minimised.

Other Heuristics There are some other ways to simplify the pattern generation. One of them is to use the so-called level packing, which packs items in a line at the same level. When an item cannot fit, it creates a new level on top of the current highest item, and puts the rest of the items on this new level; and so on. For such a variant, many heuristics of 1D problems can be easily extended and applied. Surveys by Csirik et al. [49] and Coffman et al. [41] cover many such heuristics: worst case ratios for Next Fit (NF) and Next Fit Decreasing Height (NFDH) are 2, and for First Fit (FF) and First Fit Decreasing Height (FFDH) are 17/10.

2.6 Meta-heuristics

Meta-heuristics is a generic name for the class of optimization algorithms that iteratively search for better solutions. The basic idea that underpins most meta-heuristics is based on the fact that, although solutions of a problem might be multi-modal and/or discrete, they normally exhibit some degree of 'similarity' between each other. The relationship enables us to arrange the solutions into a structure called a neighbourhood. An existing solution can be altered within the structure by a step or a move to another point in its neighbourhood. Normally, the bigger a step the less similarity there is between the old and new solutions. This provides a foundation of many search algorithms. Different strategies control the move in the solution space, deciding whether to accept the new solution or not, to maximum the chance we find the optima.

A simple search strategy is local search, which includes First Descent (FD) and Best Descent (BD, also known as steepest descent). In first-descent, an existing solution will be replaced by the first, if any, better solution; while in the latter, all neighbouring solutions will be enumerated and the best will be chosen as new candidate, if it is better. Compared to FD, BD usually has a better solution quality at the cost of more evaluations. An issue with simple local search algorithms is, when the search space is discrete and multi-modal, they tend to become trapped at a local optima, i.e. no neighbouring solutions are better than the current one, even though a non-neighbouring solution might be. In addition, the move of local search is not adaptable, the algorithm has no ability to learn from the search experience; it's also a single-thread search which may not be efficient enough for large search spaces.

To avoid being trapped at a local optima, more sophisticated approaches have been introduced such as Variable Neighbourhood Search (VNS) [99, 100], Simulated Annealing (SA) [124] and Tabu Search (TS) [85]. VNS is a more systematic local search which constructs a hierarchical neighbourhood around an existing solution and searches iteratively from the nearest to farther neighbourhoods until a better solution is found and substitutes the current one. Simulated Annealing (SA), Tabu Search (TS) accept worse solutions strategically with the hope of escaping from the local optima. SA simulates the physical annealing process in metallurgy, in which atoms actively change initial positions when the temperature is high and gradually settles down as the temperature is decreased. SA uses the temperature as a controlling parameter, initially being set very high to encourage diversification in the search space by allowing many worse solutions to be accepted. As the search progresses the temperature is lowered to allow more focused exploration. TS applies a more explicit strategy that maintains a list of moves which will be forbade for a certain number of iterations.

Evolutionary Algorithms (EA), include Genetic Algorithm (GA), Genetic Pro-

gramming (GP), Evolution Strategies (ES) etc., are more sophisticated search paradigms, which were first formalized in early 1960s but have flourished only since the 1970's due to the advance of more powerful computer technology which made their application practical. The most notable characteristic of EA approaches is that they search in parallel with the notion of population for a set of candidate solutions. With a selection mechanism based on the fitness of each solution. The fitter candidates in the population have higher chances of being selected for reproduction, which mimics the idea behind Darwinian natural selection. The evolution of one population after another also provides a basis for learning the structure of the search space. Many recent developments are using the algorithms to build models of a problem and use the models to direct the search into promising areas. In the next two sub-sections we briefly review the basic notions of Genetic Algorithms, Evolution Strategies and their applications on cutting and packing problems, which provides the foundation for our research.

GAs, GPs and ESs maintains a population of candidate solutions, reproduce new solutions through variation processes and exert selection pressure to guide the search. There are significant differences between these two algorithms. GAs search in the space of symbolic encoding (genotype), while ESs normally process phenotypes directly. In the reproduction and variation processes, GAs rely more on recombination of multiple existing solutions (parents) while ESs tend to use mutation. The theory behind GAs are largely based on schema theorem and building block hypothesis, while ESs find its root in asymptote properties of gradient landscape and Markov Probability. GPs differ from GAs in that the population consists of specialised individuals of computer programs (or functions) traditionally implemented with a tree structure.

2.6.1 Genetic Algorithms (GA)

The representation of GAs are typically encoded as strings of alphabets with finite length, called chromosomes. The alphabets can be binary or ordinal numbers. More complicated chromosomes can also be constructed by elementary strings, such as those in messy-GA or Grouping GA (see chapter 4), where the primary strings represent shapes to pack and additional ancillary strings can be used to indicate bins that each shape belongs to.

Algorithm 1 illustrates the framework of the canonical GA. The core steps of the algorithm are the inner loop from line 4 to line 6. With the selection mechanism (line 4), candidates with higher fitness value are more likely to be chosen for reproduction. In conjunction with the recombination step (line 5), good genetic encoding will be inherited and passed to the next generation. This is the idea of 'survival of the fittest' that hopefully converges the search process to good regions of the search space. There are many options for the selection and crossover operators. Selection methods that are most commonly found in the literatures include roulette wheel selection, tournament selection, and truncate selection, etc. [168]. De Jong [54] compares various selection methods by an empirical study, which reveals the relationship between convergence speed and selection pressure. For the crossover of binary strings, there are one-point, multi-point and uniform crossover are used, while for ordinal number strings, order-based crossover, partial matching crossover have been specially designed to guarantee feasibility on certain problem types. Researchers also recognise the importance of mutation (line 6) as an operator that provides random small variation to existing solutions. The benefit of such variation is to prevent premature convergence, i.e. the population becomes stuck at a local optima with homogeneous genetic encodings. In the merge step (line 9), there are also two main types of strategies depending on if the best candidates from parent generation are kept. If they are, the GA are normally referred as the steady state GA.

Algorithm 1 Pseudo-code for the canonical genetic algorithm

```
1: initialize population
2: while not meet stop criteria do
3:    while children less than new pop_size do
4:       select two parents from population;
5:       crossover to generate two children;
6:       mutate the two children;
7:       decode and evaluate children;
8:    end while
9:    merge populations;
10: end while
```

Theoretical research on GAs has largely focused on the schemata theory [106] and the building block hypothesis [87], which attempt to understand the dynamics of GAs evolution process. A schema is a template of partial genetic encoding, which can be matched to a sub-set of candidate solutions' encodings. The average fitness of these candidates provides a measurement of the fitness of the schema. The schemata theorem also gives rise to the Building Block (BB) Hypothesis. Bridges and Goldberg [86] show that short, low-order schemas, the BBs, with above average fitness will dominate the population, as they will increase exponentially in subsequent generations. Some variants of GAs are trying to identify and exchange BBs in more effective ways. In real applications many problems are BB-hard problems, i.e. BBs are hard to find, highly interactive or easily disrupted [67]. The extreme is the phenomenon of deceptive functions [88]. That is, BBs which are not 'fit' in themselves, are in fact necessary in generating fitter solutions. Another problem we will demonstrate in chapter 4 is that, while explicit search for BBs can aggressively direct the search to promising paths, the number of BBs might increase exponentially for instances of larger sizes. This side effect could eventually tip the performance scale if not controlled properly.

2.6.2 Evolution Strategies (ES)

The general procedure of ESs resemble that of GAs in many steps (see Algorithm 2). In most implementations, both algorithms randomly generate the initial solutions as the starting point of the search (line 1), utilise the populations to search in parallel (lines 3 to 6) and select good candidates based on their fitness (lines 4 and 8). However, GAs use crossover/recombination operator as the main reproduction method and mutation as an assisting variation operator, while ESs places greater emphasis on mutation. As such both have very different underlying theories [16].

Algorithm 2 Pseudo-code for the canonical evolution strategies

```
1: initialize population
2: while not meet stop criteria do
3:    while number of new children less than pop size do
4:       select a parent from population based on fitness;
5:       clone and mutate to generate children;
6:       adjust mutation strength;
7:    end while
8:    merge populations based on fitness;
9: end while
```

2.6.3 Application on One Dimensional Problems

Meta-heuristics have been applied to cutting and packing problems as they produce good quality solutions thanks to the increasing power of modern computers. For one dimensional problems, as the representation of a problem is much simpler than higher dimensional problem, standard meta-heuristics can often be applied, while in the high dimension scenario, many of these approaches use a hybrid strategy combining a meta-heuristic together with a placement heuristic. For example, in a typical GA method, the GA searches for the best permutation of shapes that enables a decoder to return a good packing solution. In addition, many hybrid meta-heuristics are also devised in the literature to form stronger strategy for certain problems, such as Memetic Algorithms combining Evolutionary Algorithms

with Local Search can often improve the results of many problems. In this section, we review application of these algorithms on one dimensional problems, more complex and hybrid applications on higher dimensional cases will be introduced in the next section.

Local Search (LS) In [155] the neighbourhood is simply defined as exchanging up to p items in one bin and q items in another. In addition to two typical search methods, Best Improvement (BI) and First Improvement (FI), another search method, Priority Improvement (PI), is presented as follows: Step 1 sorts bins according to a specified priority so that bin i has the ith highest priority. Step 2 for $m = 3$ to $2N - 1$ (N is the number of bins), searches the neighbourhoods between all pairs of bins i and j ($i < j$) such that $i + j = m$.

Shouraki and Haffari [171] investigated the fitness landscape of one dimensional Bin Packing Problems, they applied the STAGE search [113], a variant of Local Search, to the problems and compared it with steepest descent, first descent and stochastic hill climbing. The STAGE algorithm uses a learning strategy to construct predictive evaluation functions rather than the static objective function to guide search.

Simulated Annealing (SA) Foerster and Wäscher [75] tackled a variant of one dimensional stock cutting problem which has items in batches of orders and the objective function is to minimise the maximum span of any batch. The SA they designed features a standard Boltzmann function $e^{(\frac{-\delta}{T})}$ for acceptance probability, a decreasing temperature with coefficient of 0.75 and an increasing number of search loops with coefficient of 1.15 after each move. The solution quality is equivalent to 3-opt but using much less computation time.

Genetic Algorithms (GA) Early implementations of Genetic Algorithms usu-

ally use binary string encodings. However, this approach encounters some intrinsic difficulties for cutting and packing problems [164]. In [164] Reeves pointed out that another encoding method called q-ARY has some defects, though the method improves binary encoding to some extent. According to his studies hybrid GAs with some simple on-line heuristics, notably FF, BF and NF, are promising approaches. Some combination of reduction procedures and linear programming are also compared in this paper.

The current state of art shows that two other approaches are promising. The first, Grouping GA (GGA), was introduced by Falkenauer [66, 67]. The other, combining GA with an online decoding heuristic, will be discussed in a later section as it is mainly used for 2D packing problems. GGA is designed to handle the so-called problems of redundancy and disruption of schema in GAs. Instead of single string chromosome, each has an additional labelling string. Genes are said to be in the same group if they have the same label indicators. Genetic operators such as crossover and mutation work on a group basis. A novel fitness function is also proposed in [66]. Bhatia and Basu [17] modified GGA slightly and proposed a multi-chromosomal encoding for bin packing problems.

Ant Colony (AC) In recent years, ant systems have drawn a lot of interest from researchers. In [132], Levine and Ducatelle hybridised Ant System with the FFD pack heuristic and a local search improvement by Martello and Toth [150]. Their approach encodes the pheromone $\tau(i, j)$ as the favourability of having item of size i and j in the same bin. The pheromone will be reinforced if two items appear in the same bin produces a good solution. In [21], the pheromone matrix is defined as one row for each item size and one column for each remaining space. A new pheromone updating function and fitness function can be found in the paper.

2.6.4 Application on Multi-dimensional Problems

A comparison of various meta-heuristic approaches SA, GA, TS can be found in [108, 109]. All these approaches utilised the search space of permutations of shapes. Burke and Kendall [26] also compared the three meta-heuristics but on a simplified testing problem which has identical shapes and aims to minimise the bounding box.

Local Search (LS) Faroe et al.[68] used similar representation and neighbourhood structure as the simulated annealing in [55] (see below), but adopted a different Fast Local Search strategy to avoid the slow convergence issue in [55].

In [111, 112], Imahori et al. formulated a two dimensional rectangle packing as a permutation of sequence pair (δ_+, δ_-) for each shape. Three neighbourhood moves were defined in the paper which swap two shapes' sequence pair or shift a shape's sequence pair in either or both dimensions.

Simulated Annealing (SA) SA has been studied by many researchers. In [55], a neighbourhood structure is defined, by a relax-and-penalise strategy, as shifting items in both x and y dimensions, and the objective function is to minimize the overlapping area. Searching proceeds by allowing temporary infeasible overlapping, with the overlap being punished through the objective function. Lai and Chan [130] represented a SA that searches in the symbolic representation space, which permutes the orders of shapes to be cut. Fainal [65] extended SA to work with both guillotine and non-guillotine cutting patterns. Burke et al. [29] created a hybrid two-stage strategy which packs a sub-set of shapes initially with the Best-Fit [28] deterministic heuristic followed by a SA search stage permuting orders of the rest of unpacked shapes (rotation of shapes is allowed in the work). Sokea and Bingul [172] proposed the combination of SA and an improved bottom left fill heuristic.

They also examined the effects of the cool schedule and other parameter settings of SA.

Tabu Search (TS) Alvarez-Valdes et al. [2] applied tabu search to non-guillotine cutting problems where new solutions were derived by altering shapes adjacent to waste areas and inserting new shapes. The tabu list was used to keep from a non-improving move that had similar wasted area (the similarity is defined by the minimal virtual rectangle covering all wasted area).

Lodi et al. [137, 141] presented a tabu search using the neighbourhood search that tries to rearrange a subset of items into a target bin. The algorithm always accepts improved solutions; it also accepts solutions not in the tabu list but having equal fitness value to incumbent solutions; for deteriorated solutions a penalty function was used to determine if a move is accepted. The tabu list was designed to prevent repeating the last certain number of moves performed.

In an unpublished report [47] Crainic et al. proposed a tabu search algorithm based on the interval graph representation [74].

Genetic Algorithms (GA) Jakobs [114] used a GA for more general packing patterns, including irregular shapes, with generic chromosomes corresponding to orders in which shapes were placed into the bins. A Bottom Left (BL) packing heuristic is applied to map a genetic encoding to a packing. The order-based encoding was also applied to three dimensional problems in [120]. Gomez and Fuente [90, 91] adopted exactly the same encoding as Jakobs'. They also compared different crossover schemes Partial Matching Crossover (PMX), Order Crossover (OX) and Cycle Crossover (CX) for such an order-based encoding. An enhancement of the ordered list encoding was proposed by Dowsland et al. [58], which showed the benefit of incorporating bounding information of wasted areas when each shape was added. Liu and

Teng [134] also improved the standard approach by utilising an improved BL decoder assigning downward movement with priority and a more sophisticated fitness function favouring less fragmented unpacked area. Babu and Babu [5] extended Jakobs' algorithm to multiple-sheets stock cutting problem by prefix the chromosome with an index list of sheets used.

Many other special encodings can also be found in the literature. Kröger [126] applied a GA to guillotinable packing problems. He proposed the concept of a hierarchical structure of meta-rectangles (a tree structure representing the relationship of guillotine cuts) to construct a packing, which ensures the guillotine pattern at each step and reduces the complexity of problems. The slicing trees were encoded as a string of alphabet $1, 2, \ldots, n$ for n shapes and v, h for cutting orientation. Kröger [127, 128] also tried modifying directed graph representation of a packing, where two arcs t-edge and r-edge indicate "on-top" and "to-right" relationship, by deleting either one of the t-edge and r-edge for each shape and resulting in a single-edge directed binary tree encoding. Special operators of mutation and recombination were introduced to generate new solutions.

Bortfeldt [18] designed for 2D container loading a complex layer structure as in [19]. The expression of solutions describes layouts explicitly thus does not need decoding heuristics. Khoo et al. [122] presented a tree-like encoding system, which simulates the packing from a bottom left corner towards the top right hand side. In [153] Mukhacheva and Mukhacheva showed how to use a GA for semi-infinite strip packing which utilises a unique representation of block structure. Instead of ordinal numbers in [114], Goncalves [92] composed a chromosome with n random keys uniformly distributed between 0 and 1 and used an extra step to translate each chromosome into a Rectangle Packing Sequence before calling a decoding heuristic.

Some real world applications can be found in the following publications. [133] and [110] dealt with three dimensional and non-convex problems. Herbert and Dowsland [104] tackled pallet loading problems where all shapes are identical and rotation is allowed. They experimented both one-dimension binary string and two-dimension binary matrix as representation which preserves the notion of closeness of positions on the pallet. Bortfeldt and Gehring [19] tackled container loading problem with a GA that represent the problem with complex layer data structure.

Hybrid Algorithms A tabu search algorithm, hybridised with a parametric neighbourhood search strategy, is presented in [136–138, 141]. The neighbourhood search acts in a way which rearranges items in k different bins in an incumbent solution by some heuristics. The parameter k controls the neighbourhood size, and increases by one if no improving solution is found or a solution is prohibited by the tabu list. The parametric neighbourhood search strategy balances the search diversification and intensification by adjusting the value of k.

Another hybrid local search algorithms can be found [111, 112]. The authors examined different relationship types between pair wise items. Based on the relationship a local neighbourhood is constructed and searched.

Genetic Programming (GP) An innovative Genetic Programming approach is proposed by Burke et al. in [25]. GPs are usually regarded as a meta-heuristic, in particular, a variant of GA. However, the approach of Burke et al. searches for better evaluation functions instead of evolving physical packings, therefore it can be classified as a hyper-heuristic and will be discussed in the next section 2.7.

2.7 Hyper-heuristics

As previous sections have explained, exact algorithms and heuristics are often tailor-made algorithms and often limited to specific classes of problems. Meta-heuristics although more versatile still require in depth knowledge on both problem domains and low level heuristics' properties [44]. This is one of the motivations behind hyper-heuristics which aim to deliver highly adaptable 'off-the-shelf' optimisation tools for wider range of problem domains [27, 165].

Early development of hyper-heuristics were focused on the idiom of 'heuristics to choose heuristics', which includes using greedy strategies [42, 43, 156], or other meta-heuristics, such as Simulated Annealing, Tabu Search. Some advanced learning strategies from other areas cross fertilise the research in hyper-heuristic. In [167] a learning classify system incorporated reinforcement learning into hyper-heuristic framework. These hyper-heuristics are utilising low-level heuristics that are pre-determined (only high-level strategies are adaptable). The motivation behind these hyper-heuristics are, given that the different decoders and parameter sets perform differently, it normally relies on a user's experience (or even intuition) to make appropriate choices. Being aware of the difficulties faced by heuristic and meta-heuristics, a natural question to ask is if we can develop an automated system which requires less human interaction and can still deal with a wide range of problems, i.e. can we raise the generality of 'black-box' type of algorithms?

One possible way is presented for one-dimensional bin packing problems by Ross et al. [166, 167]. In their approaches they associate a set of packing heuristics with different packing states. A learning classifier system[167] and a genetic algorithm[166], act as higher level managers, looking for an appropriate packing heuristic to be employed at each step of the packing. In these approaches, a set of predefined problem states is used to describe the states of partially filled bins and the remaining pieces. In [167], a classifier system determines the problem

states, therefore, the low level heuristic to call. This approach requires a good understanding between problems and low level heuristics. For many situations, such as in high dimensional cases, the task of gaining such understanding and enumerating all possible situations can be non-trivial, thus we have to resort to different techniques. In [166], a GA is employed to detect problem states and suitable heuristics to call. As we demonstrate in the next section, some heuristics such as left-justify or right-justify, may not be relevant to problem states, but are still crucial in some cases to produce good solutions.

More recently, a new approach emerged which uses the idea of 'heuristics to generate heuristics'. In [22, 25] Burke et al. applied Genetic Programming (GP) to evolve a population of functions as packing heuristics. Each individual function is represented in a tree structure comprising of arithmetic operators and size measures as terminal nodes. Unlike other hyper-heuristics introduced above, the GP is generating packing heuristics rather than searching for an existing heuristic to match a packing state. The ability of the GP to generate new heuristics from elementary building material (i.e. terminal nodes) is important for the hyper-heurisitc to ahcieve the goal of self-adaptiveness and generalisation. The Local Search approach by Shouraki and Haffari [171] (introduced in 2.6.3) also adopts a learning strategy to construct predictive evaluation functions.

A good introductory paper of hyper-heuristics can be found in [23]. More recent developments can be found in [30] and [34]. Hyper-heuristics have also been applied to other areas such as time-tabling [24, 162], and SAT [6, 22].

2.8 Summary

This chapter has reviewed models and solution approaches of cutting and packing problems in basic one dimensional and higher dimensional context. There are various mathematical programming models which can solve instances of small

sizes to optimality. Many techniques have been reported in the literature for solving Linear Programming and Integer Programming models, including column generation, branch and bound and etc. Heuristic is another type of model, which can usually find good solutions with less computational cost even for large size problems. Researches on heuristics are interested in the best, worst and average performance analysis. More sophisticated approaches like meta-heuristics have been created to iteratively apply simple heuristics and search for improvements on solutions. One of the recent developments on meta-heuristics is the approach of hyper-heuristics, which does not directly search in solution space, but via an indirect search in the space of combination of multiple heuristic decoders. Hyper-heuristics have been shown to be effective on many practical problems, however the theory of hyper-heuristics is still an area needs more investigation. In the next chapter, we investigate a hyper-heuristic for multi-dimensional orthogonal packing and show the approach can search wider solution space than traditional meta-heuristic approaches.

Chapter 3

Multiple Low Level Heuristics Hyper-heuristic Approach

3.1 Introduction

In Chapter 2 we reviewed the standard heuristics and meta-heuristics which are commonly found in the scientific literature to tackle NP-hard problems, including orthogonal packing problems. There are some concerns regarding these approaches. Firstly, Coffman et al.[40] pointed out that the performance of one heuristic, in terms of both worst-case and average-case, may vary depending on given instances. Their proofs presume a uniform distribution of item sizes and are applicable to one dimensional problems. Performance for other distributions and higher dimensional instances are much less understood. The lack of insight into instance properties and heuristic behaviour causes difficulty in practical situations when we need to select an appropriate heuristic for the problem at hand. Secondly, a lot of heuristics are designed to guarantee feasible packings, especially in high dimensional cases. These heuristics may be so biased that they might not be able to construct certain patterns, effectively stopping us being able to find the optimal solution[11]. The limitation is also inherited by any meta-heuristic which

employs only one of these heuristics.

Hyper-heuristics are an emerging search methodology that are motivated by the goal of raising the level of generality of search methods to overcome the difficulties encountered by standard (meta-)heuristics. The approach has been successfully applied to many problems [23, 27, 38, 165]. In this chapter, we propose a genetic algorithm based hyper-heuristic approach, which is able to overcome the bias of just using one heuristic and is able to search a larger solution space without loss of efficiency. The hyper-heuristic intelligently chooses a suitable heuristic each time we need to place an item, which enables it to achieve a higher level of generality by operating well across a wide range of problem instances. There are two main differences from previous, standard genetic algorithm approaches. Firstly, a set of heuristics will be utilised, so as to avoid any shortcomings in only using one. Secondly, the chromosome in our hyper-heuristic framework encompasses both heuristic information as well as which item to pack. Therefore, in our approach we enhance the standard GA encoding, where a chromosome is a permutation of items, by adding a set of heuristics together with probabilistic information. Such information will facilitate choice decisions in selecting heuristics rather than relying on a user's arbitrary judgement. A learning mechanism is responsible for updating the probabilities according to the historical performance of the heuristics we employ, thereby influencing future behaviour. The aim is to build a black-box system that can raise the level of generalisation at which the algorithm can perform on this class of problems.

While there is no formal definition of hyper-heuristics, they can generally be described as heuristics (or meta-heuristics) to choose heuristics (or meta-heuristics) [27]. Burke et al. [23, 27] and Ross [165] introduced a conceptual framework which suggests a segregation of the roles between the high level search and low level heuristics. The higher level acts as an overarching search strategy but instead of searching through a problem space the search operates on a set of heuristics,

where each heuristic transforms the underlying problem in some way. That is, the hyper-heuristic searches through heuristic space, rather than, more traditionally, searching through the problem space. Nevertheless, the precise dynamics of high level and low level (meta-)heuristics is still not fully understood. Therefore we propose a refined framework for our particular approach, which helps explain the behaviour of our method. We draw on further evidence to support our conclusions by testing our methodology on two-dimensional orthogonal strip packing problems, and comparing our results against standard GA approaches.

The rest of the chapter is structured as follows. Section 3.2 presents our hyper-heuristic approach. Section 3.4 reports our results on benchmark instances. Section 3.5 present our conclusions and suggestions for future work.

3.2 The GA-based hyper-heuristic approach

3.2.1 Overview

Our hyper-heuristic approach is based on a genetic algorithm (Algorithm 3). However, it differs from standard GAs by utilising a set of decoding heuristics at the lower level instead of only one. To facilitate the choice of heuristics, the standard chromosomes are enhanced by combining the order of rectangles with heuristic-probability pairs. Section 3.2.2 gives more details on the chromosomes. Compared to the hyper-heuristic approach using static learning classifier system[166, 167], our approach adopts a more dynamic roulette-wheel selection mechanism to choose an incumbent heuristic from the candidate set (line 11 in Algorithm 3), along with an adaptive learning mechanism to intelligently recognise the more suitable heuristics within a set (lines 15 to 19 in Algorithm 3).

g_max is the maximum generation in search usually determined by the computational time allowed. In each generation, the algorithm generates a set of S solutions. For individual packs the set R of items with probabilistic choice of

Algorithm 3 Pseudo-code of the GA-based hyper-heuristic framework

1: set generation counter $g = 0$;
2: **for** $x = 1$ to population size **do**
3: permute shapes by random shuffle $\mathbf{R} = \{r_1, r_2, ..., r_n\}$;
4: for rectangle $r_i \in \mathbf{R}$ initialize a set of heuristic-probability pairs (h_i^j, p_i^j);
5: evaluate s_x and insert into population S;
6: **end for**
7: **while** $g++ \leq g_{max}$ **do**
8: select parents $s_x, s_y \in S$;
9: generate new child s_c by crossover and mutation of s_x, s_y;
10: **for** $i = 1$ to n **do**
11: choose a heuristic h_i^j according to probability p_i^j;
12: pack r_i with h_i^j;
13: **end for**
14: s_c to child set S_c;
15: set $\Delta = (Height_{s_x} - Height_{s_c})/Height_{s_x}$;
16: $p_i^j = p_i^j + \Delta$;
17: **for** $j' \in \mathbf{J} \setminus j$ **do**
18: update $p_i^{j'}$;
19: **end for**
20: merge S_c, S into S;
21: **end while**
22: return best solution $s_{best} \in S$.

heuristics from the heuristic set of J. r_i, h_i^j and p_i^j denote the ith rectangle and the associated jth heuristic with the probability p_i^j of being used.

Given the many heuristic decoders reviewed in chapter 2, an immediate question we need to answer is how many, and what type of heuristics, are to be included for selection. We will use empirical experiments to investigate the proper size and type of the heuristic set. In addition, we compare two alternative versions of the hyper-heuristic to decide the types of heuristic decoders:

1. **Non-competing heuristic sets (NC-HH):** The types of heuristics are fixed. They are all available to pack each shape no matter how low the probability of one being called might turn out to be (see lines 7 to 9 in Algorithm 4). In this version, although the heuristics h_i^j are arbitrarily chosen and remain static, the probabilities p_i^j are updated adaptively and the search procedure is still a dynamic probability selection mechanism. Algorithm 4

shows details of the refined procedures for this version of the hyper-heuristic.

Algorithm 4 Refined pseudo-code for NC-HH

```
1: //refined initialising (replaces line 4 in Algorithm 3)
2: for i ∈ {1, 2, ..., n} do
3:    initialize a set of |J| heuristics, and set each p_i^j = 1/|J|;
4: end for
5:
6: //refined updating (replaces lines 17 to 19 in Algorithm 3)
7: for j' ∈ J \ j do
8:    p_i^j' = p_i^j' − Δ/(|J|−1);
9: end for
```

2. **Competing heuristic sets (C-HH):** This hyper-heuristic chooses initial heuristic sets, and it allows badly performing heuristics to be replaced (Algorithm 5). When initializing, the hyper-heuristic randomly selects a sub-set of heuristics from all those available. During the updating process, if the probability of a heuristic drops below a threshold level, it will be replaced by another randomly chosen heuristic. Whenever replacement happens, the probabilities of the heuristics will be reset to allow the newly introduced heuristic a fair chance of competing with the surviving members that are already in the set. All heuristics will be assign a probability of $1/|J|$. In effect, all heuristics are competing against each other in order to stay in the candidate set.

3.2.2 Chromosomes

We enhance the standard genetic algorithms' chromosome structure by including with each item (allele) some probabilistic information for heuristic selection. Each allele is denoted as a set J of heuristics and probabilities h_i^j and probability p_i^j, $i = 1, 2, ..., n$. n represents the number of items to be packed and j is a parameter defining the number of candidate decoding heuristics available to each rectangle. Figure 3.1 shows a chromosome for the proposed hyper-heuristic methodology. $r_1, r_2, \cdots, r_n$ is a permutation of n items, which defines a packing order. Attached

Algorithm 5 Refined pseudo-code for C-HH

1: //refined initialising (replaces line 4 in Algorithm 3)
2: **for** $i \in \{1, 2, ..., n\}$ **do**
3: random select a set of |J'| < |J| heuristics, and set each $p_i^j = \frac{1}{|J'|}$;
4: **end for**
5:
6: //refined updating (replaces lines 17 to 19 in Algorithm 3)
7: **if** the incumbent heuristic h_i^j has $p_i^j < Prob_{th}$ **then**
8: replace h_i^j with a new heuristic, set $p_i^j = \frac{1}{|J'|}$;
9: reset other non-incumbent heuristics' probabilities $J \setminus j$;
10: **else**
11: **for** $j' \in J \setminus j$ **do**
12: $p_i^{j'} = p_i^{j'} - \frac{\Delta}{|J|-1}$;
13: **end for**
14: **end if**

to each r_i there is a set of heuristics and probability pairs. p_i^j determines the probability that heuristic h_i^j associated with piece i will be applied, and $\sum_j p_i^j = 1$ for each i.

r_1	r_2	...	r_n
(h_1^1, p_1^1)	(h_2^1, p_2^1)	...	(h_n^1, p_n^1)
(h_1^2, p_1^2)	(h_2^2, p_2^2)	...	(h_n^2, p_n^2)
...	...	...	...

Figure 3.1: A hyper-heuristic GA chromosome

The probabilities (initially set equal) will be updated through a learning mechanism. The choosing of a heuristic for each piece is considered as an action that will be rewarded or punished according to the results of the final packing height, i.e. the probabilities of incumbent heuristics will be increased if we obtain a better packing, and decreased otherwise. These changes are made in proportion to the changes of the height of children compared to their parents. Therefore, the whole system learns from its interaction with the search problem. For example, a system may find it tends to apply rules finding lower position for large pieces, while for small pieces there is less difference in heuristic probabilities. This is a critical difference between hyper-heuristics and other meta-heuristics, as the hyper-heuristic utilises this adaptive policy to learn how to choose lower level heuristics.

3.2.3 Decoding heuristics

Decoding heuristics for higher dimensional problems are concerned with two decision problems: which space to select for the placement and where in the chosen space to place the item. For the first question, we will use three categories of heuristics: First Fit, Next Fit and Best Fit. These heuristics were initially defined in the literature for one-dimensional problems [40]. For higher dimensional problems there are several difficulties. The partial packing usually contains non-convex spaces. Some modelling methods [35, 63] divide such a space into several overlapping sub-spaces. A consequence is that packing a piece will sometimes affect several available sub-spaces, which is different to one-dimensional bin packing where available spaces are independent of one another. We will next describe a general data structure which can be used to extend them to higher dimensional problems. For the second question our hyper-heuristic will consider all four corners of a chosen space. Therefore, in our experiments, we have twelve different placement options (and thus heuristics) for each item. As we will present in the results section, the type and quantity of heuristics will affect the performance of the hyper-heuristic (possibly because the larger the size of the pool, the potentially higher computational overhead and the larger search space). Therefore, we limit the candidate sets to a more manageable size rather than using all twelve heuristics. The data structure is a list of all available spaces L_a, which is similar to the implementation of Hayek et al. [63]. It divides a non-convex empty space into a finite set of enclosed rectangle sub-spaces. We introduce how to update the list L_a, which initially contains only the empty container, when adding shapes. More details can be found in Section 5.3.1, where both insertion and remove operators on packings are explained.For example, in Figure 3.2 the original empty space can be split into 4 rectangle sub-spaces. Note that some of the sub-spaces may overlap.

When inserting another shape, the placement may affect several overlapping

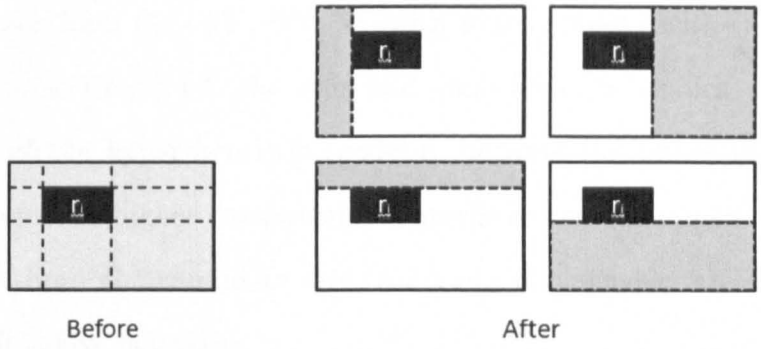


Figure 3.2: Changes on available spaces (in grey) before and after placement of r_i (in black)

sub-spaces. For each of the affected sub-spaces, the placement will further split it into at most four smaller sub-spaces. Note, if packing an item in a corner, it will only create at most two new sub-spaces. Therefore the complexity of the splitting is at worst $O(2^n)$. In practical situations it is normally much less complex because not all sub-spaces will be affected by one placement step, and moreover, the splitting process usually generates many redundant sub-spaces, which are completely enclosed in another larger sub-space and therefore can be removed from the list at the end of the procedure. As long as we have a complete list of all available sub-spaces, we can then define:

1. **First Fit (FF):** select the feasible sub-space at the lowest level, break ties by choosing the left most sub-space (equivalent to bottom-up heuristic by Baker et al. [11]);

2. **Best Fit (BF):** select the feasible space with the smallest area;

3. **Next Fit (NF):** sub-spaces not exposed from the top of the partial packing will be removed from the list (result in a shorter list than FF and BF), then select the lowest feasible sub-space (equivalent to bottom-left move with downward priority by Liu and Teng [134]).

For the second decision our hyper-heuristic will consider all four corners of a chosen space. Therefore, in our experiments, we have twelve different placement

options for each item (i.e. FF, BF, NF with four corners each). As we find by experiments in section 3.4.4, the type and quantity of heuristics will affect the performance of the hyper-heuristic (possibly because the larger the size of the pool, the potentially higher computational overhead and the larger search space). Therefore, we limit the candidate sets to a more manageable size of four rather than using all twelve heuristics.

3.2.4 Selection and replacement strategy

Parent selection is carried out by an elitist strategy and roulette-wheel selection. By experiments we found that it is more effective to select parents from the top third, rather than the entire population, according to fitness values, when selecting those for reproduction. These parents will generate the same number of children as the population size. A child chromosome will replace the worst one in the population if it has better fitness value and is not a replica of an existing chromosome in population. The purpose here is to ensure convergence, while also maintaining a certain degree of diversity in population.

3.2.5 Recombination

The GA operators are standard, involving a random two-point order-based crossover (2OX) [168] and mutation for reproduction of populations. Figure 3.3 and Figure 3.4 show how 2OX and mutation operate. The detailed settings of parameters will be introduced in the following sections. In particular, when exchanging orders of items in a sequence the associated set of heuristics of each item will be inherited. We have implemented two other operators, partial matching crossover (PMX) and single point crossover (1X), which also guarantee feasibility. Compared with 2OX, PMX makes little difference and 1X performs slightly worse.

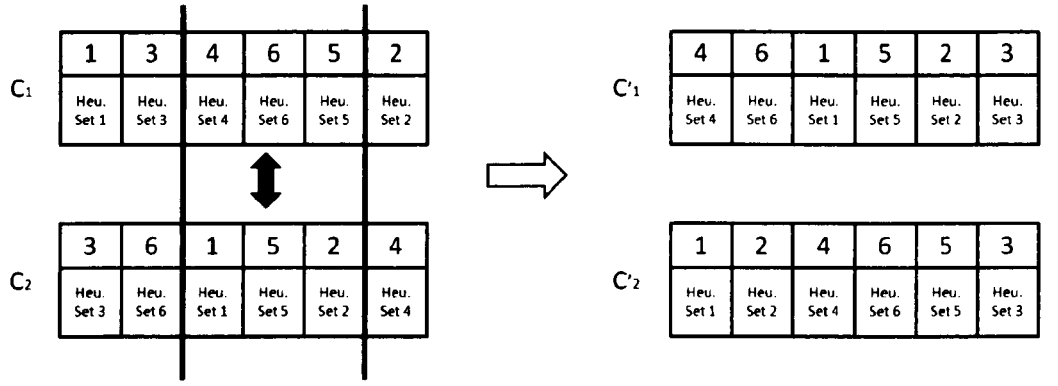


Figure 3.3: GA 2-point crossover

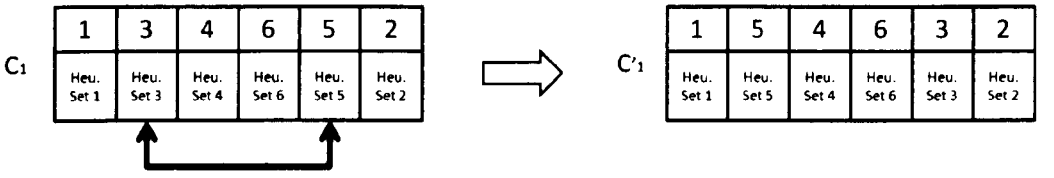


Figure 3.4: GA Mutation

3.3 The hyper-heuristic framework

In a standard GA, an individual in the search space X is evaluated by a single mapping function h' to the solution space Y (Figure 3.5). h' is a deterministic mapping process and it creates a many-to-one relationship from the search domain to solution space. In our proposed hyper-heuristic approach, we have the same space of chromosomes which still has $n!$ permutations. However, in the hyper-heuristic scheme each permutation is mapped through a set of heuristic functions which forms an intermediate heuristic space (Figure 3.6). If we have j different heuristics and n shapes, since each shape will probabilistically choose one out of j heuristics to pack, there will be j^n different permutations of packing heuristic sequences. For example, for any specific shape permutation, we can apply $(h_1^1, h_2^2, h_3^3, \cdots)$ or $(h_1^3, h_2^2, h_3^1, \cdots)$. It is obvious that standard GAs are simply special cases of the hyper-heuristic approach where we apply a single heuristic $(h_1', h_2', h_3', \cdots)$ for all shapes. It is easy to see the solution space of standard GAs are only subsets of the hyper-heuristic solution space. Therefore, a direct conse-

quence of this approach is that the solution space is enlarged as the new strategy overcomes the weakness of a single heuristic mapping. The new solution space includes the original solution space, since solution space Y can still be achieved by applying the same heuristic at each step of the packing. Thus the hyper-heuristic has the ability to find more solutions without losing the original solution space. Another observation is that the set of evaluation functions are not equally treated. By the learning procedure each function derives a different probability $\prod_{i \in n} p_i^j$ of being called, which leads to the selection of "fit" functions. Finally a possible effect is that as each function has a chance of being applied, the result will be robust in terms of a lower standard deviation. These two hypotheses are still subject to further investigation, but we will use empirical evidence to support these claims.

3.4 Experimental results

The purpose of this empirical study is to examine the effectiveness of the proposed hyper-heuristic, i.e. exploration of a wider solution space, average results, consistency in terms of standard deviation and speed of convergence. It is also interesting to investigate the impact of the size of the heuristic sets, which is an important parameter affecting the size of search space.

The benchmark instances are taken from Burke et al.[28] and the OR-library (`http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/`). C1 to C7 are seven categories with three instances in each and N1a to N7e are 35 non-guillotine instances. N1 to N12 have the number of items ranging from 10 to 500, and the other two sets of instances have 16 to 197 items. We have also created a set of instances to show that hyper-heuristics can explore a wider solution space.

The algorithm was implemented in C++ and ran on a grid computer with 2.2GHz CPUs, 2GB memory and GCC compiler. To obtain statistics on average and standard deviation, every experiment has been run 100 times. Note the

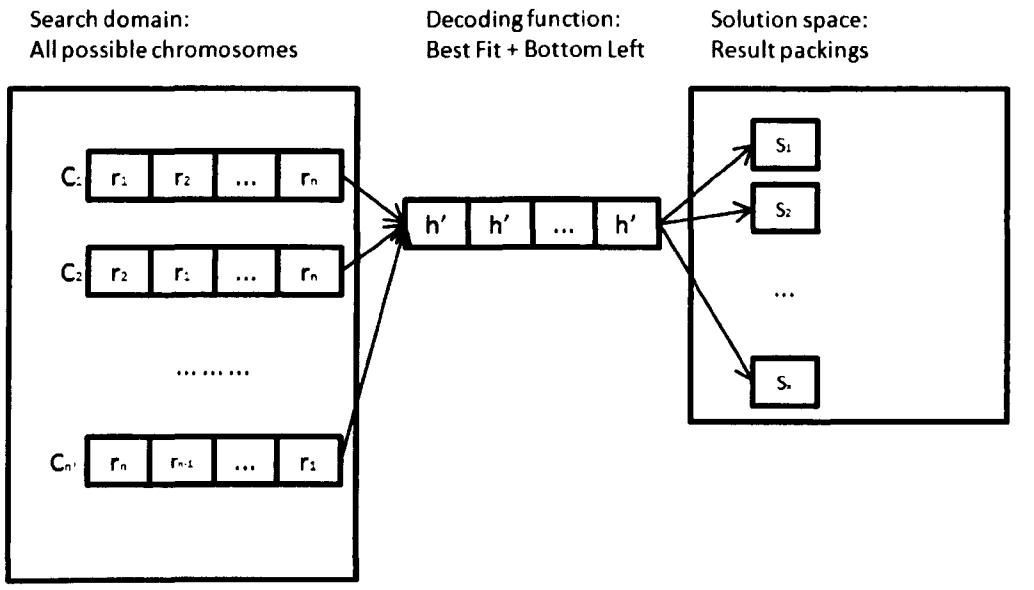


Figure 3.5: standard GA framework

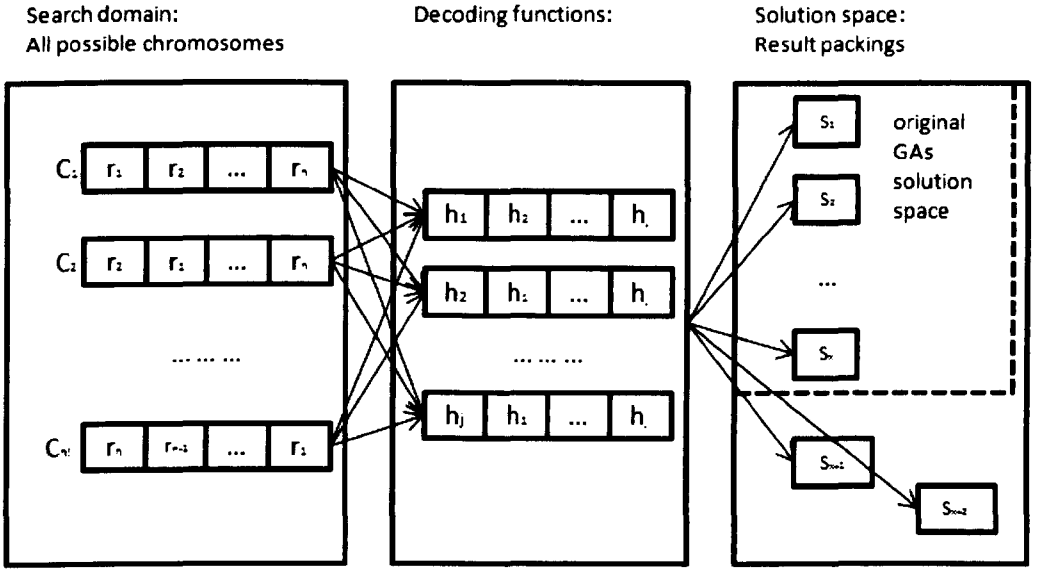


Figure 3.6: Hyper-heuristic framework

computational costs for all algorithms compared in this section are based on the total number of evaluations rather than time, as counting evaluations provides a more objective measure for theoretical purpose which exclude factors such as the programming skills and quality of testing machines. For practical use, running time is however an important matter, therefore we also provide more information from our experiment where appropriate.

3.4.1 Feasibility and optimality

The first set of experiments is designed to evaluate the effects of multiple decoders. We created some instances where gaps exist in the middle of patterns in the optimal solutions (as per Baker et al. [11]). Using only one heuristic will fail to achieve the optimal pattern. An example of such an instance is as follows. Nine Items: 60x60, 60x60, 50x50, 50x50, 40x40, 40x40, 10x10, 10x10, 31x30 to be packed into a strip of width of 151. (Note if the last item has size of 30x30 and the strip has a width of 150, the shapes would fit perfectly in the strip.) The best results achieved by meta-heuristic with a single placement heuristic (in our experiments GA+NFBL (GA with Next Fit and Bottom Left Fill), GA+FFBL (GA with First Fit and Bottom Left Fill), GA+BFBL (GA with Best Fit and Bottom Left Fill)) and hyper-heuristics (both C-HH and NC-HH versions) are 120 and 110 respectively Figure 3.7). It is simple to verify that 110 is the optimal. Assuming the optimal is less than 110, say 109, the whole area of strip needed (including any utilised and wasted areas) is 16,459 (151x109), which is less than the total area of all items 16,530, therefore it is impossible.

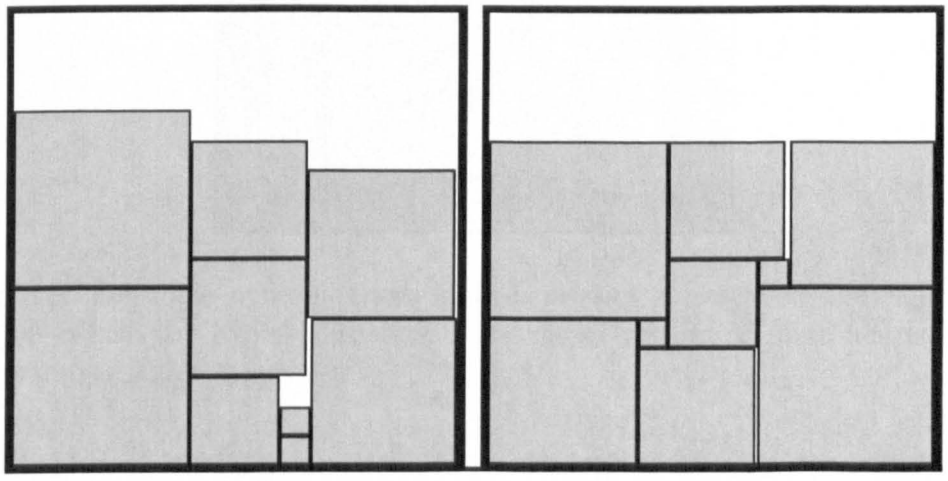


Figure 3.7: Best result achieved by meta-heuristic is 120 and optimal achieved by hyper-heuristic is 110

The experiments provide evidence that hyper-heuristics can avoid the drawbacks of applying only a single heuristic, and find more feasible solutions and,

possibly, optimal solutions.

Other instances in our new dataset are created by choosing a number of pieces and cutting at random points (see Appendix A for the dataset and Table 3.1 for the experimental results). By cutting the original pieces in this way, we consequently create some smaller pieces that may be used to fill the gap in the middle of the pattern. For example, although there are still small gaps, they can effectively be shifted to the boundaries of the pattern, i.e. in some cases optimal solutions can then be achieved by applying a single heuristic, such as the pattern in Figure 3.8, which can be created by the bottom-left, or equivalent, heuristics. However, even if a single heuristic can, in theory, find the optimal solution for the class of problems created in this chapter, the hyper-heuristics in our experiment still demonstrate stronger performance, as shown in the next section.

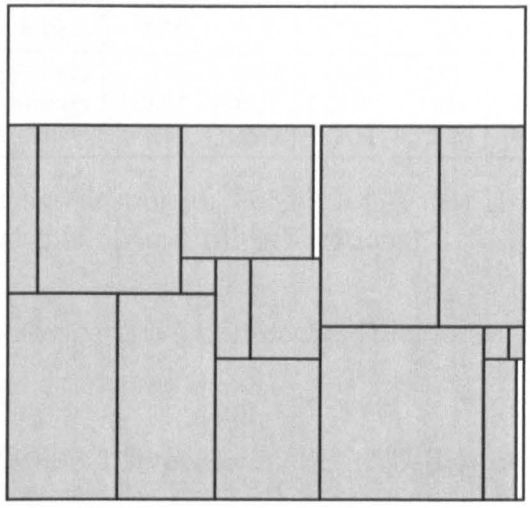


Figure 3.8: For some new instances meta-heuristics can achieve the same best-result of 110 as the hyper-heuristics, but average-results of meta-heuristics are still worse (see Table 3.1)

3.4.2 Performance and consistency

The proposed hyper-heuristic is searching for a solution indirectly through calls to heuristics (section 3.3). In experiments presented in this section we compare

		Next Fit		First Fit		Best Fit	
		HH	GA	HH	GA	HH	GA
instance1	min	110	120	110	120	110	120
	average	112.3	120	110.1	120	110	120
	st dev.	4.23	0	1	0	0	0
instance2	min	110	120	110	120	110	120
	average	119.3	120	118.1	120	116.3	120
	st dev.	2.56	0	3.94	0	4.85	0
instance3	min	110	110	110	110	110	110
	average	113.04	113.7	111.85	112.43	111.77	111.91
	st dev.	0.93	1.42	1.52	1.56	1.51	1.65
instance4	min	110	120	110	120	110	110
	average	120.8	120.7	120	120.1	119.5	119.6
	st dev.	3.39	2.56	2.46	1	2.19	1.97
instance5	min	111	111	111	111	111	111
	average	115.26	115.45	113.1	114.31	112.43	113.75
	st dev.	1.55	1.73	2.04	2.16	1.81	2.35
instance6	min	120	120	120	120	120	110
	average	121	120.5	120	120.2	120.2	120.1
	st dev.	3.02	2.19	0	1.41	1.41	1.74
instance7	min	112	112	110	111	110	111
	average	114.3	115.06	113.26	114.2	112.61	113.42
	st dev.	1.16	1.46	1.22	1.49	1.21	1.38
instance8	min	116	117	116	116	116	114
	average	119.67	120.37	117.62	118.4	118.5	118.82
	st dev.	1.44	1.89	1.11	1.29	1.3	1.4

Table 3.1: Results for new instances. For both GA and HH total evaluations are 5,000 (at population size of 50 and 100 generations)

our hyper-heuristic to standard GA approaches [108] for each category of decoders (FF, NF and BF).

In Table 3.2 and Table 3.3, hyper-heuristics (NC-HH) utilise four corners while the GA only uses bottom left positioning. By looking at the average and the deviation we find that the hyper-heuristic performs equally well as standard methods (producing superior solutions in more cases on the First Fit and Best Fit but slightly worse solutions on Next Fit). Particularly, for the new class of instances we created according to Baker et al. [11], hyper-heuristics are superior.

dataset	no. of instances	First Fit			Next Fit			Best Fit		
		HH win	GA win	equal	HH win	GA win	equal	HH win	GA win	equal
n1-n12	12	5	5	2	2	9	1	9	2	1
c1-c7	21	12	7	2	8	12	1	11	10	0
n1a-n7e	35	16	19	0	21	14	0	18	17	0
new	8	8	0	0	6	2	0	7	1	0
total	76	41	31	4	37	37	2	45	30	1

Table 3.2: Average over 100 runs

dataset	no. of instances	First Fit			Next Fit			Best Fit		
		HH win	GA win	equal	HH win	GA win	equal	HH win	GA win	equal
n1-n12	12	5	6	1	5	6	1	8	2	2
c1-c7	21	13	7	1	7	13	1	9	11	1
n1a-n7e	35	17	18	0	16	19	0	17	16	2
new	8	5	3	0	3	5	0	5	2	1
total	76	40	34	2	31	43	2	39	31	6

Table 3.3: Standard deviation over 100 runs

3.4.3 Convergence

We also carried out a further test to explore the convergence properties of the proposed algorithm. For each problem, we plot the average results obtained from 10 to 300 generations with a step size of 10 (with a fixed population of 50). The actual run time for both NC-HH and C-HH are around 2% higher than GA, which reflects the slight overhead of computation of choosing heuristics. Figure 3.9 gives examples of the non-competing hyper-heuristic (NC-HH) on two instances (same results observed on all other instances), from which we can see clearly that the hyper-heuristic converges as quickly as a standard GA. This result suggests that the increase in search space can be compensated by the effectiveness of the hyper-heuristic.

3.4.4 Effects of the number of heuristics in a set

The above experiments show the effectiveness of hyper-heuristics. However, as we explained in section 3.3, the greater the number of heuristics we make available to the hyper-heuristic, the larger the search space of the algorithm. In the next set of experiments we attempt to find a suitable trade-off between the size of

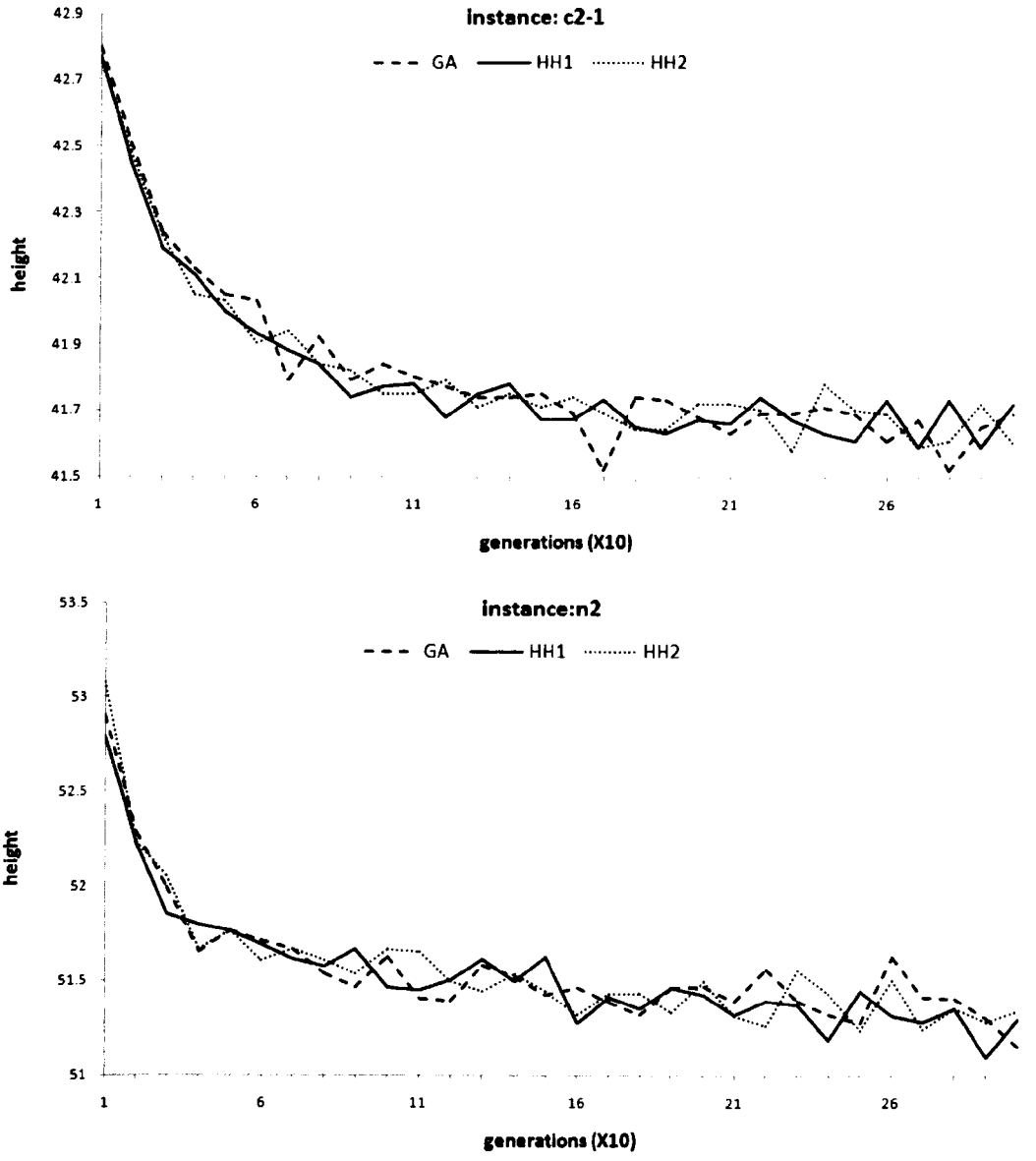


Figure 3.9: Average converging over 10 to 300 generations

the set (and thus computational time) and solution quality. In Table 3.4, we present a comparison between four runs of a hyper-heuristic (C-HH version) where heuristics are all randomly chosen and the set size varies between 4 and 8. It can be seen that many of the best results (highlighted) are produced with just four heuristics, while hyper-heuristics with five or more decoding heuristics are superior on less instances. The experiment supports our hypothesis that increasing the number of low-level heuristics makes the search algorithm more effective by

combining the strength of multiple heuristics. The experiments also show there is a trade-off between the effectiveness of multiple heuristics and the increased search complexity. Interestingly, for larger instances such as n11 and n12 (300 and 500 shapes respectively), where the search spaces are already much larger, the negative effects of increasing search complexity seem not as significant as in smaller instances, where the hyper-heuristics with six and seven decoders outperformed the ones with less decoders. Therefore, we recommend employing four heuristics for small and mid-sized problems, while for larger problems more heuristics can be beneficial.

3.4.5 Comparison with other methods

In this chapter we want to evaluate the effectiveness of a GA-based hyper-heuristic with the primary goal to show the advantages of multiple decoders over a single decoder. In the literature there are other specially designed hybrid methods that perform particularly well on cutting and packing problems. Table 3.5 shows two other approaches, GRASP [3] and a hybrid method of Best-fit with Decreasing Width (BFDW) [28]. However, many of these methods may have difficulty in solving certain classes of problems when only a single heuristic is used. On the other hand, the hyper-heuristic's framework is flexible, and we could adopt GRASP as high-level search strategy and/or BFDW as one of the lower-level decoders. We believe by combining other meta-heuristics search operators and heuristics decoders, the results could be further improved. It is worth noting that the results from the hyper-heuristic approach are robust as indicated by the smaller standard deviation values in Table 3.5, when compared to the other approaches. Burke et al. [27] pointed out hyper-heuristic approaches have the potential to be utilised for a much wider set of domains which is not the case with bespoke systems developed for a given domain.

Instance	set size 4	set size 5	set size 6	set size 7
n1	40	40	40	40
n2	51.51	51.32	**51.08**	51.22
n3	**52.59**	52.61	52.65	52.66
n4	**84.08**	84.29	84.39	84.4
n5	**106.19**	106.42	106.5	106.42
n6	104.78	104.63	104.7	**104.55**
n7	**110.04**	110.37	110.2	110.26
n8	**85.79**	86.26	86.02	86.35
n9	**156.48**	156.61	156.7	156.63
n10	**154.36**	154.66	154.8	154.64
n11	155.88	155.97	155.8	**155.75**
n12	317.26	317.28	317.2	**317.13**
C1-1	20.02	**20**	20.01	**20**
C1-2	**21.19**	21.24	21.21	21.2
C1-3	20.09	**20.04**	20.05	20.08
C2-1	**41.64**	41.74	41.81	41.67
C2-2	41.35	41.44	**41.33**	41.4
C2-3	40.97	**40.94**	40.96	40.95
C3-1	63.23	**63.22**	63.27	63.22
C3-2	**63.14**	63.3	63.3	63.3
C3-3	**63.34**	63.38	63.41	63.4
C4-1	**64.72**	64.95	64.88	64.98
C4-2	**64.59**	64.77	64.83	64.88
C4-3	**64.13**	64.34	64.37	64.41
C5-1	**64.41**	64.68	64.71	64.72
C5-2	**65.12**	65.55	65.49	65.51
C5-3	**64.58**	64.72	64.71	64.79
C6-1	**86.05**	86.42	86.48	86.48
C6-2	**87.11**	87.41	87.5	87.33
C6-3	**86.21**	86.52	86.55	86.53
C7-1	**173.41**	173.99	174.1	174.13
C7-2	**172.29**	173.1	173.2	173.05
C7-3	**173.65**	173.7	174.8	174.54

Table 3.4: Heuristic set size affects results

3.5 Summary

In this chapter we have demonstrated an improvement over standard genetic algorithms by adopting a hyper-heuristic framework to tackle NP-hard problems, in particular packing problems. The idea is to combine a set of heuristic decoders

Instance	Opt	Burke GA + BF	Alvarez-Valdes GRASP	NC-HH	C-HH
n1	40	40	40	40	40
n2	50	50	50	50	50
n3	50	52	51	51	51
n4	80	83	81	83	83
n5	100	104	102	104	104
n6	100	102	101	103	103
n7	100	104	101	104	104
n8	80	82	81	83	84
n9	150	152	151	154	154
n10	150	152	151	152	152
n11	150	153	151	154	154
n12	300	306	303	315	314

Table 3.5: Compare with other algorithms

with a high level search operator. The hyper-heuristic is able to raise the generality of an algorithm by overcoming the drawbacks of just employing a single heuristic. Both analysis of the algorithm's framework and empirical studies demonstrated that the hyper-heuristic approach works well and is certainly worthy of further investigation. The potential benefits can be summarised as follows:

- Compared to standard approaches the hyper-heuristic is able to explore a larger solution space. Therefore, it has the potential to find the global optima or better results than some other standard meta-heuristics.

- Its built-in learning mechanism is highly automated requiring less user judgement, as all suitable lower-level heuristics can be put in a set, and the hyper-heuristic itself will choose from that set. It is also flexible for further expansion by having the option to add new heuristics into the candidate set.

- Our empirical study shows that, by selecting appropriate parameters (in our case the size of heuristic set), the hyper-heuristic is able to converge at the same rate as more traditional meta-heuristics, and also to perform more consistently.

In this chapter we have only investigated standard GA as a search operator and a relatively small number of heuristics as decoders. There is space for improvement by integrating other (meta-)heuristics into the hyper-heuristic framework, such as GRASP[3]. However, there is further work required to understand the dynamics among different level of operators. It is especially interesting to observe the interaction and trade-off between the intelligent evolutionary sampling and the more complex search space.

The hyper-heuristics proposed in this chapter belongs to the family of Evolutionary Algorithm (EA). While for the hyper-heuristics we are interested in the learning mechanism for the selection of decoding heuristics, other aspects of EAs, including representation and neighborhood search strategies, have significant effects on performance. In the next chapters we will investigate these important aspects of EAs and further improve their performance on the packing problems. In Chapter 4 we will continue to investigate a GA-based algorithm, which has an even more sophisticated chromosome representation and explicitly enhance the building block hypothesis of GAs. In Chapter 5 we will explore a hybrid genotypic and phenotypic representation.

Chapter 4

Dynamic Grouping Genetic Algorithm

4.1 Introduction

In the previous chapter we improved a standard GA by utilising a set of decoding heuristics rather than just relying on one. In this chapter, we focus our attention on a number of fundamental issues surrounding GAs, especially the representation, the recombination mechanism and the theoretical basis of the Schema Theorems and the Building Block Hypothesis (BBH). In particular, this chapter further investigates an important extension of genetic algorithms, the Grouping Genetic Algorithm (GGA), for solving cutting and packing problems. The algorithm was first proposed by Falkenauer [66] for one-dimensional bin packing. GGAs use more complex genotype representations than simple GAs and search for good partial solutions which are then recombined to build new solutions. The idea is to explicitly enhance the building blocks (BBs) implied in simple GAs.

This chapter makes contributions on both EA theory and the application of EAs on Orthognal Packing Problems. Firstly, from an algorithmic perspective, we proposed a building-block network model to study the evolution of the algorithm

which can help to explain the strength and weakness of the algorithm. Based on Holland's Schema Theorems, the BBH was first proposed by Goldberg et al. [87] but it received heavy criticism from other researchers [181]. By analysing the network model, as well as showing evidence from the experimental results, we show the 'double-sided' effects of BBs. On one side the BBs helps recombination to be an effective operator, and accelerate the search process, but on the negative side the number of BBs will increase quickly as the instance size gets larger.

From an application perspective, we further extend the GGA to more general orthogonal packing problems, including higher dimensional problems and single container problems (strip packing), for which grouping is more complex. The algorithm developed in this chapter has been able to solve benchmark instances to optimality for instance size up to 40 items, while previous best results found in literature are only up to 20 items.

Section 4.2 introduces a model for multi-dimensional packing problems. The model decomposes solutions into a hierarchical network of partial solutions. It is used to describe the relationship among groups of shapes as partial solutions, and one way to utilise generation-wise information of evolutionary computation to help selection be more effective. It provides a foundation for the GGA implemented in Section 4.3. The details of the implementation include how the GGA detects good partial solutions which are dense areas meeting some prescribed criteria, and how to select compatible partial solutions to reproduce for the next generation. Results for this approach are reported in section 4.4. While the simple form of a GGA shows its strength in some of the instances, the limitation of this approach is that the number of good partial solutions increases exponentially, and the scalability deteriorates quickly. Potential improvement approaches are discussed in section 4.3. One of the approaches using a static grouping technique will be presented in Chapter 5.

4.2 Group Network Model

The GGA developed in this chapter uses a different definition for groups compared to both the one used in the Grouping Evolutionary Strategy (GES) in Chapter 5 and the one by Falkenauer [66]. The GGA introduced by Falkenauer was for one dimensional bin packing problems and items are naturally grouped by bins. In higher dimensional packing, a shape's location affects its neighbors' location in all dimensions simultaneously, while in one dimensional problems a group size is a simple summation of all member shapes. For strip packing problems, there is only one container so it is not possible to group shapes by containers. Therefore, the definition of a group has to be generalized for higher dimensional packing or strip packing problems.

In this chapter, a group for a multi-dimensional packing is defined as a subset of shapes $\mathbf{S} \subseteq \mathbf{R}$ enclosed by minimal orthotope $\mathbf{Q}$, where shape $r \in \mathbf{S}$ if and only if r is enclosed by $\mathbf{Q}$. Let $|\mathbf{S}|$ be the size (or order) of a group, which is the number of its shape elements. Let $V : r \rightarrow \Re$ denote a volume function, i.e. v_i is the volume of i. The density of a group is then

$$\frac{\sum_{i \in \mathbf{S}} v_i}{v_Q} \in [\theta, 1],$$

and $\theta \in [0, 1]$ is a pre-defined threshold for a 'good' group.

The relationship among groups can be illustrated by a hierarchical network as shown in Figure 4.1. A node in the network represents a group. A directed edge $\langle S_i, S_j \rangle$ means S_i appears in S_j as a subgroup (therefore the order of S_i is less than S_j, i.e. $|S_i| < |S_j|$). Nodes are arranged in layers from left to right with increasing size. For example, nodes in the first layer (leftmost) are all individual shapes of an instance; the next layer on the right contains groups of two shapes that can be built from layer one; and the rightmost layer are complete solutions containing

all shapes. Notice that some directed edges may not go from one layer to the next but "jump" to a few layers to the right, meaning one group may appear as a subgroup in another group of several orders larger, $|S_i| - |S_j| > 1$. Group S_i and S_j are compatible groups when they share no common shapes, i.e. $S_j \bigcap S_i = \emptyset$.

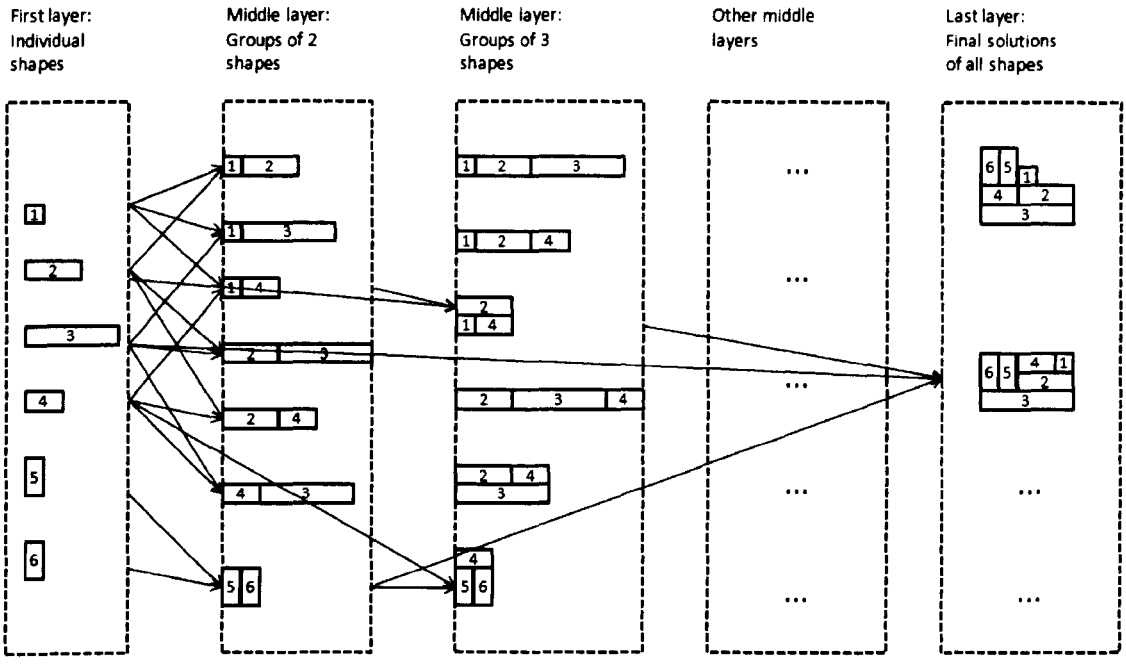


Figure 4.1: Hierarchical network of groups of a packing problem (some arcs and packings omitted for clarity)

The network describes the relationship among partial solutions. Although having an incremental nature, it is, however, not a model for an online packing process as many heuristics are. As explained before, a node may jump, meaning the it does not necessarily align with another shape to make a larger group but it may align with other groups containing a few shapes to transform to a new group of several orders larger. Moreover, even if a node increases its order by one i.e. aligned with only one other shape, this newly added shape does not necessarily come immediately behind, but maybe after many other shapes in the packing sequence. In essence, the model captures the information of the final location of each shape rather than consider permutations of packing sequences. It is a desirable feature since redundancy is avoided and network size is drastically

reduced. The threshold θ can also help us control the number of nodes in the network model. For example, when $\theta = 1$, we only look at perfectly packed partial solutions which have no wasted area.

Another feature of the network model is its ability to incorporate statistical information when nodes transform from low order to high order. Given a group S_i, if we uniformly randomly select a fixed number of groups from compatible groups, a transition matrix can be calculated to describe the probability for a node to transform to a larger node. If we maintain a record of nodes identified, not only within a generation, but for several generations, this probability matrix will be able to reflect the generation-wise evolution information on how larger groups are gradually formed during a search process.

There are some limitations of this network model. First, although the number of nodes is less than the number of permutations of each subset of the shapes, it can still increase exponentially as instance size increases. In such a scenario, it is not possible to derive a complete transition matrix as the table would be too large to compute. Some techniques will be needed to limit the nodes to a more tractable size, such as those used in Bayesian Optimization Algorithms (BOA) [157]. Another limitation is for a non-guillotine packing instance where no subset of shapes can form a group with density above a θ value. For example, in Figure 4.2, no subset of shapes can form a no-waste partial packing, i.e. any subset of shapes is a concave having a very low value of θ. In such a case, the network will not provide much useful information to guide the search. How to model the packing process for these instances is an interesting open question. For the OPP investigated in this thesis we do not enforce special patterns (such as guillotine pattern) and the network model is applicable.

The network model is consistent with the building block hypothesis, since the nodes can be regarded as explicit building blocks. The model provides the foundation for the GGA introduced in the next section. At the beginning of the search we

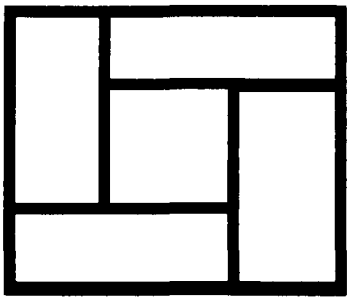


Figure 4.2: 2D non-guillotine packing

initialize the population, some groups of low orders are likely to be detected and inserted as new nodes into the left side of the network. During subsequent search, nodes of higher order are gradually found, mainly by recombining smaller building blocks, towards left side of the network. The mutation operator may introduce some new building blocks that have been undetected before. The replacement strategy used in the GGA approach is not to replace an entire generation, but only to replace the worst members. This means, in the network, we retain some building blocks found in previous generations, which are usually smaller on average. These small blocks are necessary to complement large blocks in building up further blocks. The threshold θ is a control parameter and would be tuned for different instances. For guillotine instances, there are paths made up with perfect nodes, zero waste partial solutions, from left to right in the network. We would like the search to be biased towards such paths, so we can set θ to 1. For instances of free form packing, θ tends to be less than 1 to allow more exploration of not so perfect partial solutions. One difficulty for the GGA is the possibility of exponentially increasing the number of building blocks. Some remedies can be conjectured from analysis of the search process within the network model, which will be explained towards the end of this chapter.

4.3 Implementation

4.3.1 Overview

This section presents the implementation details of the GGA. The overall framework of this approach is shown in the pseudo-code (Algorithm 6) and flow chart (Figure 4.3). There are two important differences between this approach and simple GAs. First, chromosomes in the GGA have varied length with each allele being a single shape or a group of shapes. While in standard GAs, each allele is an individual shape, thus the chromosomes are made up of these alleles and have equal length. Secondly, besides the common steps as in standard GAs (lines 3, 4, 5 and 9 for selection, recombination, decoding, replacement), there are a few extra steps (lines 6, 7 and 8) in the GGA search loop. In line 6, after all shapes being packed, solutions will be examined to see if any new groups have been created. Once identified, the new groups will be used to transform the encoding before its being merged back into population (line 7). The algorithm also moves groups to a tabu list, if any of the groups has been re-used many times but are not able to improve the results(line 8).

Algorithm 6 Pseudo-code for grouping genetic algorithm

```
1: initialize population
2: while not meet stop criteria do
3:    select two parents from population;
4:    generate two children;
5:    decode children;
6:    identify groups;
7:    transform encoding with identified groups;
8:    move unfit groups to tabu list;
9:    merge populations;
10: end while
```

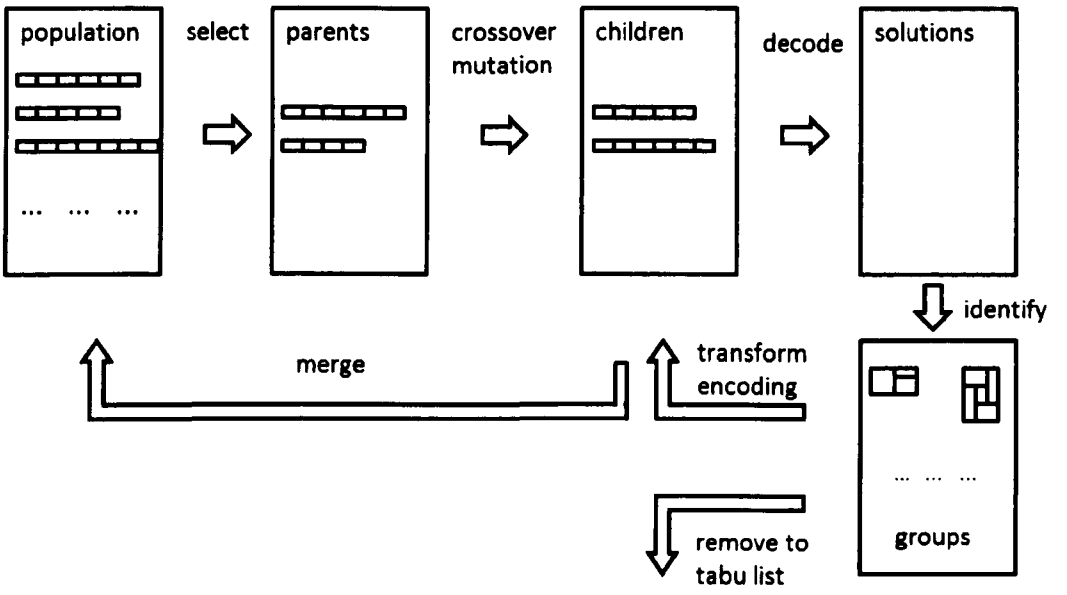
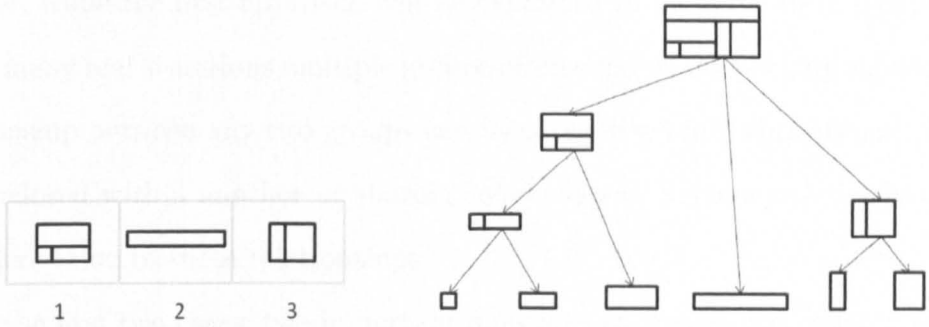


Figure 4.3: Flow chart of the search loop in Grouping Genetic Algorithm

4.3.2 Chromosome

In standard GAs for packing problems, chromosomes are most commonly encoded as linear structures [109], e.g. vector, list, and the alleles are individual shapes to be packed. The structure can be modified to suit GGAs by allowing alleles to be groups of shapes as in Figure 4.4(a). This domain-independent structure has a strength that it can be used for many other problem domains such as the Traveling Salesman Problem (TSP), and scheduling, etc. Many operators, including crossover and mutation, are easily applied in order to generate new populations. For cutting and packing problems, due to the Euclidean geometric property, another potential choice is to use tree structures as shown in Figure 4.4(b), which is a more natural hierarchical representation of the grouping relationship. Leaf nodes correspond to individual shapes. Some leaf nodes can make up groups represented by non-leaf nodes, which may recursively construct further larger groups up to the root node, representing the final packing layout. If this tree structure is adopted, tree operators such as those in Genetic Programming (GP) can be used [125]. In this chapter, we implement the linear structure (Figure 4.4(a)) as it is

more versatile, so that the algorithm developed here can be further derived into other algorithms or transferred to other problem domains.



(a) List Chromosome (b) Tree Chromosome

Figure 4.4: Two forms of GGA chromosomes

When initializing the population, a chromosome is simply a random permutation of all shapes. After packing, and in the genetic encoding transform step, if some shapes can be grouped together, they will be removed from the sequence and the group will be inserted as a single allele at the location where the first member of the group appeared in the original sequence. Some complexities in composing and transforming the genetic encoding are explained below (see Figure 4.5).

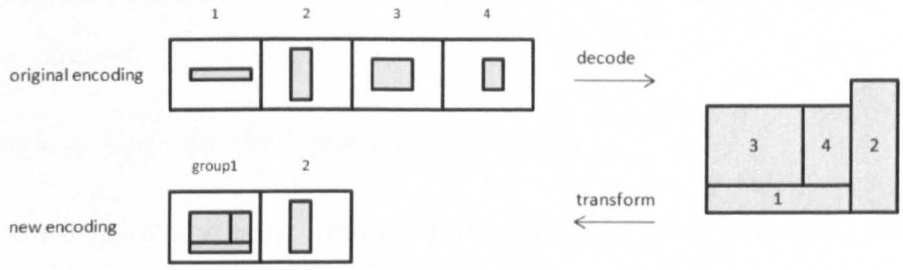


Figure 4.5: Grouping Genetic Algorithm transform original encoding to new encoding

The first issue, when grouping shapes into a block, is if the density θ is less than 1 it will allow gaps or "dents" in the block. The block is then no longer an orthotope as the original shapes were, but a special type of irregular concave where each edge is parallel to one of the axes. One way to handle this problem is to use the phenotype operators developed in Chapter 5 to pack such non-perfect

orthotopes. Another way is to set θ equal to 1 to force each group to be a no-waste orthotope. The latter approach will be adopted for the experiments in this chapter, while the first approach will be explained in more detail in Chapter 5.

In many real situations multiple groups often exist in one packing solution. The relationship between any two groups can be categorized into three types: disjoint, one enclosed within another or sharing some shapes. Some subtleties have to be considered due to these relationships.

In the first two cases, two important questions to answer are: when transforming the original encoding, how many groups should be used (single or multiple groups); and if a group contains some subgroups, which of them should be used (large or small groups)? It is natural to conjecture that different approaches to these two questions will cause some trade-off between search speed and quality. In one aspect, the more (or larger) groups being identified, the shorter the encodings and the smaller search space, which will usually leads to a faster search. Whereas, more individual shapes (or smaller groups) can help the search be more exploitative around local areas, since smaller pieces give us more leeway and can be used to fill gaps that would otherwise be wasted. In our experiments, three strategies will be compared:

1. single group, only the biggest group is used;
2. a greedy strategy using multiple groups with as many shapes as possible;
3. a balanced multi-group strategy depending on the stage of search. Initially it uses a greedy strategy combining as many shapes as possible, and the maximum number of groups will be used, while towards the end of the search some smaller pieces will not be grouped to allow more local search.

When groups share some subsets of shapes, one reasonable approach is to keep only one of the groups and break up the others into smaller groups until there are no conflicts. The question is, when conflict occurs, how to evaluate these

groups and decide which ones should be kept and which ones should be broken into smaller groups. Below are a number of heuristics that could be utilised:

1. favor groups with more shapes. The more pieces grouped together, the smaller the transformed instance size, and thus the smaller the search space. However, it is more likely to become trapped in a local optima;

2. favor the groups with largest total volume. Larger groups usually imply a better lower bound for the rest of shapes;

3. favor the groups with least sum of square of sizes on each dimension. This heuristic differentiates odd groups which are larger on some dimensions but smaller on others;

4. for some instances, favor groups with small sizes on particular dimensions. e.g. in strip packing where we are particularly interested in minimizing the height, we favor groups with less height;

5. use weighted average on the above measurements to make a more balanced decision. In our experiments the evaluation is based on a function weighted on two ratios: the ratio of group volume to total volume of all shapes and the ratio of group member size to instance size:

$$w_1 \times \frac{v_j}{\sum_{i \in \mathbf{R}} v_i} + w_2 \times \frac{|\mathbf{S}_j|}{|\mathbf{R}|};$$

   where w_1, w_2 are some arbitrary weights depending on preference of size or volume metrics. In our experiments, as we have no preference, both w_1, w_2 are set to 0.5;

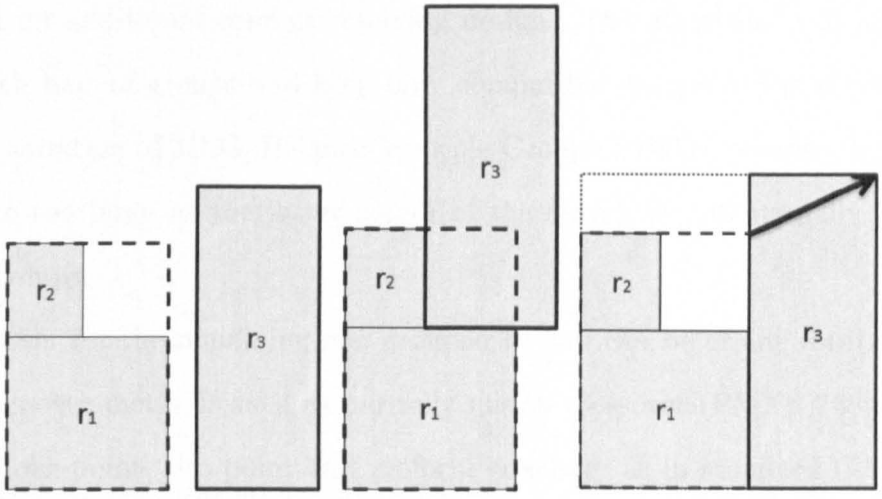
6. probabilistic choice among groups according to some of the measurements above. In our experiments, we used roulette wheel selection based on the number of elements in a group.

4.3.3 Group identification

The subroutine for identifying the groups within a packing solution needs to be efficient to avoid expensive computation. As the chromosomes are in a linear structure to mimic an online packing process, we could check for groups when each shape is added into the solution (see the pseudo-code shown in Algorithm 7). This subroutine maintains a temporary list of potential blocks, initially set to empty (line 2). When a shape is added to a solution, a new block starting with the shape will be inserted first to the temporary list (line 3). It checks the relation of the newly added shape with the rest of the blocks already in the final list (lines 4 to 14). There are three possible relations: disjoint, overlap and adjacent. When disjoint (Figure 4.6(a)), there is no effect on the final list and the temporary list. (As the GGA utilises a standard bottom left heuristic decoder, a disjoint relation implies that some shapes in between separate the block and the shape. Therefore, the new group enclosing the block and the shape is either non-perfect orthotope or a duplicate of the group containing the separating shapes.) If they overlap (Figure 4.6(b)), i.e. some part of the block is already occupied, and the occupied space exceeds the maximal allowed ratio $1-\theta$, the block will be removed from the list (lines 5 to 9). Otherwise, if they are adjacent (Figure 4.6(c)), a new block, enclosing both the shape and the block and having minimal size on each dimension, will be added to the temporary list (lines 10 to 14). After all the shapes have been packed, we remove any blocks in the temporary list with density ratio less than θ and merge to the final list.

4.3.4 Crossover and mutation

Crossover on variable length chromosomes is more complicated than for fixed length chromosomes. When groups from two parents are exchanged, conflicts may happen if any two groups share common shapes. We will examine two approaches



(a) Disjoint, no effect on existing block

(b) Overlap, delete the existing block

(c) Adjacent, enlarge the existing block (to the extent of the dotted line)

Figure 4.6: Relationships between the newly added shape r_3 and an existing potential block (the dash-lined area)

Algorithm 7 Pseudo-code for identifying groups

```
for i = 1 to n adding shape S_i do
    initialize TempList ← ∅
    TempList ← TempList ⋃ S_i
    for X ← List.First to List.Last do
        if X, S_i overlap then
            v_cumulatewaste ← v_cumulatewaste + v_overlap
            if v_cumulatewaste / v_X > 1 − θ then
                List ← List \ X
            end if
        else if X, S_i adjacent then
            NewBlock ← minimal orthotope covering X, S_i
            TempList ← TempList ⋃ NewBlock
        end if
    end for
    List ← List ⋃ TempList
end for
```

to handle the issue. The first is to avoid conflict by only using groups from one parent while carrying out crossover on the rest of the shapes. For groups inherited from a single parent, we pass the largest possible group to the children if there exists multiple ways of grouping. This strategy is named Single Largest Groups (SLG). The other approach, Multiple Largest Groups (MLG), exchanges groups

but with an additional conflict resolving design. The algorithm will iteratively check each pair of groups and keep only compatible groups in the chromosome. Another variation of MLG, Balance Multiple Groups (BMG), avoids using groups which are too large at the latter stages of the search by intentionally selecting smaller groups.

Crossover for the remaining non-grouped shapes can be of any form of order-based crossover methods such as partially match crossover (PMX) [168], or even standard one-point, two-point and uniform crossover as in standard GAs. However, in the GGA a crossover operation often leads to conflicts among groups, since two different groups may share some subset of shapes. When this happens, two strategies can be used to resolve conflicts. The first strategy is to remove the conflicting pieces from one group, which results in an orthotope with holes, and then the phenotype operators developed in Chapter 5 can be used. The second strategy is similar to the conflict resolving method used in transforming the encoding, i.e. gradually breaking up one of the conflicting groups into smaller groups until any conflicting shapes can be separated and removed from the order. In this chapter, the second strategy will be used so that we always deal with orthotope packing, and the fitness functions used to decide which group should be kept are the same as those in Section 4.3.2.

There are two different types of mutations in the GGA. One takes the form of standard GAs where two alleles swap their positions in the packing sequence. In this form of mutation each group is still treated as a whole and appear in the new solution unchanged. The other form is called re-assemble, in which a group may be selected and broken up into smaller groups, in the same way as resolving conflicts among groups during crossover. These smaller groups and shapes will then be added to the packing sequence to be re-assembled. In this form of mutation a group may only keep its partial subgroups in the new packing solution.

4.3.5 Selection and Replacement

We use the standard truncate selection with selection pressure set to 3 [54], i.e. the top third will be randomly selected to reproduce a new population whose size is equal to the size of the parent population. Different selection methods have also been tested, including tournament selection and roulette wheel selection. Their performances are similar to truncate selection. The replacement strategy adopted in the experiments is also the standard replacing-worst strategy.

In GGAs the genotype encoding is represented by groups. One motivation of this representation is to make the encoding more correlated with the evaluation of the objective function. However, good partial packings (groups) may not always generate good overall solutions. There are a few possible explanations: the interaction between groups is not linear and there may be some "bad" groups that outweigh the "good" groups; or groups slightly out of place may lead to poor packings especially when the groups get bigger at the latter stages. In our experiments, we use a combined fitness evaluation to facilitate selection:

$$w_1 \times \frac{f_g'}{\sum f_g} + w_2 \times \frac{f_o}{lb},$$

where w_1, w_2 are some arbitrary weights, f_g' is the fitness of the best group in an encoding based on chromosome fitness functions in section 4.3.2, $\sum f_g$ is the sum of fitness of the best group of each individual in the population, f_o is the value of the objective function and lb is the instance lower bound which is used to normalize the value. In our experiments we will also compare two other fitness functions: one based purely on group fitness, the other based only on objective function.

4.3.6 Hybrid strategies

Hybridization is an effective strategy in meta-heuristic search [94]. In the experiments in the next section, the GGA utilise two strategies.

Tabu list The idea of using a tabu list to exclude some local search moves is to avoid the search being trapped in a local optima. In the GGA a local optima may be caused by some groups dominating the population but those groups may not be found in the global optima. To implement the tabu list, the GGA maintains a count of how many times a group is chosen and evaluated. Once the count for a group is above a threshold, the group will be added to the tabu list and be excluded from being chosen for a number of iterations.

Restart Random restart is another technique to avoid the search being trapped in a local optima [89]. The reason for using restart can be justified with the building block network model. Suppose the initial supply of BBs are inadequate and not all initial BBs are in the global optima, the search will end up in local optima. Even if mutation breaks up existing BBs, the chances of a new BB forming, which is in the global optima, could be small. In such a scenario, restart will have more chance of forming new BBs and give them an equal opportunity in competition with other BBs. Therefore, unlike the tabu list, this approach regularly re-initializes the whole population from scratch and intends to find completely different paths to the optimal solution. The GGA uses the convergence information to decide when to restart, i.e. if there is no improvement after a certain number of generations, the population will be re-initialized. Later, we carry out an experiment that will test the frequency and draw some empirical evidence on good values for this parameter.

Initial Building Blocks (BBs) As groups are explicit BBs and are gradually built up from smaller BBs, the initial BBs are critical as they are the starting

points which greatly affect the search direction. If too few BBs are supplied there is a risk of not starting from the right point and subsequently propagating them to future generations. However when too many individuals are randomly generated, there could be too much noise as the number of ineffective BBs also increases and more evolution time could be required to converge the population. So we speculate that there is a trade-off between too few and too many initial BBs. To test the the effects of initial BBs in the GGA, we generate more random chromosomes in the first few populations to produce more initial BBs. As the search progresses random chromosomes are generated less frequently. This process is controlled by producing more random packings in the initial few generations with a probability function $1 - \frac{i}{i_m}$ where i is a counter of finished evaluations, and i_m is the maximum number of evaluations allowed by the GGA. The number of random packings in percentage of the total number of evaluations is used to indicate the relative quantity of initial BBs.

4.4 Experimental Results

Our empirical study will investigate the effectiveness of the proposed GGA. It will also explore the settings for the important parameters discussed in the previous sections. The benchmark instances are taken from Burke et al.[28] and the OR-library (`http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/`). C1 to C7 are seven categories with three instances in each, ranging from 16 to 197 shapes. n1 to n10 have between 10 and 200 items (running time for n11 and n12, containing 300 and 500 shapes, are too long, therefore these instances are not tested). The algorithm was implemented in C++ and tested on a grid computer with 2.2GHz CPUs, 2GB memory and GCC compiler. To obtain statistics on average and standard deviation, every experiment has been run 100 times. The

parameters that are common in the GGA and GAs are set to values that are mostly found in literature [54]: population size = 100, number of generations = 100, crossover rate = 0.99, mutation rate = 0.01, truncate selection with pressure of 3 and replacing only the worst.

4.4.1 Effects of θ

The first parameter of interest is the threshold value θ shown in Table 4.1. For higher θ values, the search is more biased towards non-waste orthotope. While some instances such as guillotine cases could benefit from a highly biased search, other instances may prefer lower values of θ so as to explore more of the search space. The best results are obtained when setting θ to 1 or 0.97, i.e. using groups close to zero waste. This suggests search based on less-waste groups (i.e. good partial solutions) are more likely to achieve better results, which is strong supporting evidence for the Building Block Hypothesis.

	1	0.97	0.95	0.9	0.8	0.7	0.6	0.5
n1	40	40	40	40	40	40	40	40
n2	50.00	50.00	50.05	50.12	50.01	50.23	50.17	50.24
n3	53.36	53.31	53.50	53.43	53.59	53.61	53.47	53.59
n4	84.31	84.24	84.36	84.37	84.58	84.42	84.49	84.48
n5	104.86	104.80	104.89	104.89	105.11	105.24	105.11	105.07
n6	103.77	103.72	103.88	103.77	103.90	103.92	103.93	103.94
n7	109.01	109.10	109.07	109.04	109.20	109.25	109.11	109.29
n8	85.94	86.02	85.98	86.09	86.17	86.18	86.11	85.12
n9	155.46	155.47	155.48	155.53	155.81	155.60	155.64	155.84
n10	153.65	153.61	153.70	153.68	153.78	153.86	153.78	153.83

Table 4.1: Effects of θ settings (best results of each row are highlighted in grey)

4.4.2 Effects of Recombination Strategies

Table 4.2 presents some evidence on the effects of different group recombination heuristics. The Single Largest Group (SLG) utilises the largest group identified from a single parent, while the Multiple Largest Group (MLG) strategy iteratively

finds, from both parents, the next largest group which is compatible to previously found groups. In Balance Multiple Groups (BMG) we monitor the search process and intentionally diminish the chances of large groups dominating the solutions. MLG and BMG performed equally well, while both produced better results than SLG. The result indicates the recombination of BBs is more effective than simple non-recombination approach.

	Largest Group	Multiple Groups	Balanced Groups
n1	40	40	40
n2	50.10	50.00	50.00
n3	53.48	53.36	53.42
n4	84.41	84.31	84.25
n5	104.98	104.86	104.86
n6	103.84	103.77	103.73
n7	109.07	109.01	109.01
n8	86.06	85.94	85.94
n9	155.56	155.46	155.47
n10	153.78	153.65	153.58

Table 4.2: Effects of recombination strategies

4.4.3 Effects of Fitness Functions

Apart from the grouping heuristics, we also investigated some different fitness functions of groups (Table 4.3), as introduced in the previous section. However, the experiment does not provide supportive evidence for us to favour any heuristics over the others.

4.4.4 Hybridised Strategies

The following tables (4.4 and 4.5) show the effectiveness of various hybrid strategies. As we can see the tabu list and restart strategies help to improve the GGA to some extent. A group will be moved to the tabu list (forbidding it to be used again for some time) if it has been chosen for 100 trials. The criteria for restart

	num. of elements	vol. of group	sum of sqare on dimension	weighted vol & num	probabilistic
n1	40	40	40	40	40
n2	50.00	50.00	50.00	50.00	50.00
n3	53.39	53.36	53.36	53.35	53.33
n4	84.29	84.31	84.31	84.36	84.33
n5	104.83	104.86	104.87	104.86	104.89
n6	103.77	103.77	103.80	103.78	103.80
n7	109.03	109.01	109.07	109.05	109.01
n8	85.95	85.94	85.94	85.98	85.99
n9	155.49	155.46	155.45	155.49	155.45
n10	153.68	153.65	153.66	153.68	153.67

Table 4.3: Effects of fitness functions (best results of each row are highlighted in grey)

is no improvement in 10 generations. Both tabu and restart mechanisms slightly improve the performance of the standard GGA.

	with Tabu List	without Tabu List
n1	40	40
n2	50	50
n3	52.49	53.36
n4	83.96	84.31
n5	104.88	104.86
n6	103.72	103.77
n7	108.89	109.01
n8	85.98	85.94
n9	155.49	155.46
n10	153.60	153.65

Table 4.4: Effect of tabu list

The third hybrid strategy uses a decreasing temperature to control the generation of the initial BBs (Table 4.6). We tested a set of values, the percentage represents the ratio of randomly generated chromosomes to total evaluated chromosomes. As the result shows, the initial number of BBs can make a difference to the results. In our test 10% and 15% give us the best results.

	restart	no restart
n1	40	40
n2	50	50
n3	53.36	53.36
n4	84.11	84.31
n5	104.87	104.86
n6	103.72	103.77
n7	109.00	109.01
n8	85.93	85.94
n9	155.47	155.46
n10	153.63	153.65

Table 4.5: Effect of restart

	5%	10%	15%	20%	25%
n1	40	40	40	40	40
n2	50.00	50.00	50.00	50.04	50.05
n3	53.37	53.36	53.37	53.43	53.38
n4	84.35	84.31	84.30	84.35	84.35
n5	104.84	104.86	104.85	104.95	104.92
n6	103.78	103.77	103.73	103.84	103.82
n7	109.05	109.01	109.01	109.07	109.08
n8	85.94	85.94	85.92	86.00	86.02
n9	155.49	155.46	155.46	155.51	155.51
n10	153.67	153.65	153.64	153.70	153.69

Table 4.6: Effect of initial BBs

4.4.5 Compare with Other Algorithms

The main purpose of this chapter is to use grouping techniques to enhance standard meta-heuristics such as GAs to solve the type of NP-hard problems where elements are to be clustered into groups. As shown in Table 4.7 and 4.8, the GGA can find the optimal solution for instances up to 40 shapes, while standard GA and other methods in the literature can only find the optimal for much smaller instances. Even if standard GAs can also find optimal solutions for 20-shape instance, it only achieves this with 20% success rate, while the GGA finds the optimal in every run.

	20 items (opt. 50)		30 items (opt. 50)		40 items (opt. 80)	
	min	**% opt.**	**min**	**% opt.**	**min**	**% opt.**
GA	50	20%	51	0%	83	0%
GRASP	50	N/A	51	N/A	81	N/A
GGA	50	100%	50	18%	80	3%

Table 4.7: Compare GGA with simple GA and GRASP

However, as shown in Table 4.8, when the instance size increases the performance of the GGA deteriorates to the same level as a standard GA. The possible reason is that, as we have explained in Section 4.2, the number of groups is increasing at an exponential rate.

4.5 Summary

In this chapter, we have investigated the Grouping Genetic Algorithm, using a more complex genetic encoding and grouping techniques as an enhancement to standard GAs. The OPP is a good test example for the Building Block Hypothesis, by using GGA to explicitly trace the use of BBs. While the result shows the strength of this approach on small and middle sized instances, the algorithm is difficult to scale up to larger sized instances. The reason is that the GGA proposed in this chapter discovers groups 'on-the-fly', and the number of groups increases exponentially. There are two possible approaches to overcome the drawbacks of GGAs. One approach is to consider only the most promising groups rather than tracking all groups discovered during the evolution. However, how to evaluate and identify good groups remains an open problem. It might be helpful to apply statistical learning techniques to find good paths in the network model in section 4.2. Another way to tackle the limitation of the GGA is to use more static grouping techniques and avoid the dynamic grouping strategy. For example, we can divide shapes into only two groups, critical group for big shapes and non-critical group

No. Shapes	Instance	GA + First Fit		GA + Best Fit		Grouping GA		GRASP	
		Avg	Min	Avg	Min	Avg	Min	Avg	Min
10	n1	40	40	40	40	40	40	40	40
20	n2	51.58	50	51.26	50	50	50	50	50
30	n3	52.71	51	52.69	51	52.39	50	51	51
40	n4	84.57	83	84.24	83	83.79	80	81	81
50	n5	105.76	104	105.78	104	104.79	103	102	102
60	n6	103.63	102	103.83	103	103.55	102	101	101
70	n7	109.62	106	108.35	105	108.72	102	101	101
80	n8	85.61	84	84.99	84	85.82	84	81	81
100	n9	155.86	154	155.38	153	155.39	154	151	151
200	n10	153.74	153	153.43	152	153.42	153	151	151
16 or 17	C1	20.01	20	20	20	20	20	20	20
		21.18	20	21.18	21	21.2	21	20	20
		20.14	20	20.03	20	20	20	20	20
25	C2	16	16	16	16	15.89	15	15	15
		16	16	16	16	15.95	15	15	15
		15.96	15	15.93	15	15.17	15	15	15
28 or 29	C3	31.93	31	31.87	31	30.8	30	30	30
		32.22	31	32.1	32	31.94	31	31	31
		32.12	31	32	31	30.13	30	30	30
49	C4	64.27	63	64.26	64	63.85	62	61	61
		64.31	63	64.23	64	64.37	63	61	61
		63.8	62	63.72	63	63.69	61	61	61
73	C5	95.67	94	95.58	94	95.58	93	91	91
		96.78	95	96.79	94	96.85	94	91	91
		95.9	94	95.89	94	96.05	93	91	91
97	C6	127.58	126	127.88	126	127.45	125	121.9	121
		127.79	125	127.57	125	127.3	126	121.9	121
		127.97	126	128.23	126	128.97	126	121.9	121
196 or 197	C7	256.26	254	257.09	254	257.29	253	244	244
		253.95	251	255.08	250	254.64	254	242.9	242
		254.91	252	255.44	252	254.84	253	243	243

Table 4.8: Compare the GGA with simple GAs (implemented according to [108]) and GRASP [3] (GA and GGA are allowed to run 10,000 evaluations, GRASP is allowed to run 60 seconds as reported by the authors)

for smaller ones. To apply such a static grouping strategy would require a different representation and neighbourhood search strategy. It naturally leads us into the domain of Evolution Strategy (ES) which will be investigated in Chapter 5.

Chapter 5

Phenotype Representation and Evolution Strategy

5.1 Introduction

In this chapter we investigate various Evolution Strategies (ES) for Orthogonal Packing Problems. ESs often utilise phenotype representation as the search space and rely on mutation to perform neighborhood search, unlike genetic algorithms which use genetic encoding as a surrogate search space and crossover recombination as the main source of variation. Underlying the different implementation of ES approaches, there are significant implications on the fitness landscape when using phenotype representation and mutation search operators. Based on some properties of the ESs' fitness landscape (see section 5.7.1), we propose some basic and hybrid approaches to tackle packing problems. The ESs approaches obtain better quality results compared to most of other approaches in the literature.

This chapter is arranged as follows. In section 5.2 we present a formal definition of the phenotype representation for orthogonal packing problems. While the representation provides a direct description of solutions, an abstraction of the representation using interval graphs [72] is a useful tool to compare the similar-

ity among solutions. Associated with the phenotype representation we develop in section 5.3 specific operators for orthogonal space manipulation. The principle of the mutation operator, the most important strategy in ESs, will be introduced in section 5.4.

Section 5.5 presents the implementation details of various ESs, especially the methods for adjusting strategy parameters. We pay attention to a special case of ES, Grouping ES (GES) which, in effect, decomposes a problem and can be particularly effective for heterogeneous instances. Further to the simple approaches, we develop more advanced algorithms by hybridising the ESs with a Variable Neighborhood Search (VNS) strategy (see section 5.6.3) which takes advantage of the neighborhood structure and population-wise information during evolution. Empirical results for the algorithms will be analysed in section 5.7.

5.2 Phenotype Representation

5.2.1 Definition

In the previous two chapters, we introduced genetic algorithms which evolve permutations of shapes as a surrogate search space for packing problems. The sequence of shapes is also called a genetic encoding or genotype representation of the search problem. This metaphor to molecular's DNA is somewhat misleading and problematic. As we have shown in the last two chapters, there are some weaknesses of such genetic encoding of combinatorial problems:

Loss of information GA's reproduction operators treat all elements indifferently with its symbolic encoding (binary vector, and ordinal numbers, etc.). Critical domain specific knowledge is ignored during crossover and mutation, such as sizes of individual shapes, interaction between items, dynamics of external constraints. Although by undifferentiating items, GA's operators can

be problem domain independent, the genotype representation often causes other difficulties in search. For example, building block disruption is often observed, when the same schema may result in different fitness evaluations depending on the context.

Ineffective neighborhood structure The neighborhood structure can easily be defined for genotype representations, e.g. mutation by swapping two items, crossover by exchanging segments of encodings. However, such a neighborhood move may not correlate with objective function values, as witnessed by extreme cases of the so called deceptive objective functions, in which cases the genetic encoding provide false feedback on the correct search direction.

Dependency on mapping function GAs rely on a mapping function to translate the genetic encoding to a final objective value. As we have shown in chapter 3, a single decoding function may be insufficient to search the entire co-domain of solutions, or the quality of the decoder may affect solution quality, e.g. Next Fit heuristic for bin packing problems generates inferior results compared to Best Fit or First Fit for the majority of the benchmark instances.

Computational cost Since genetic encoding requires a decoding step before evaluation, there is extra computational cost involved in such a mapping.

Phenotype encoding is another type of representation for combinatorial optimization problems that avoids some of the problems mentioned above. A phenotype reflects the solution directly and contains all domain related information. For the packing problems, we define the phenotype representation as a set of coordinate d-tuples $L_p = \{(\cdot)_1, (\cdot)_2, \cdots, (\cdot)_n\}$, with each d-tuple $(\cdot)_{i \in \{1,2,\cdots,n\}}$ representing the position of the ith shape in d dimensional space. Without loss of

generality, a d-tuple corresponds to the position of a shape's bottom left corner. The d-tuples cannot take arbitrary values, but have to satisfy the feasibility constraints of orthogonal packing. Therefore it implicitly incorporates the size information of shapes. Moreover, given the available space (i.e. the containers' space), set L_p's complement set L_a denote all available spaces deterministically. Therefore, the phenotype representation encompasses all information regarding a specific problem: all shapes' sizes and their positions and spaces still available, and directly represent a solution. One clear benefit of such an explicit phenotype representation is that the representation is highly correlated to the objective function, i.e. the feedback information from evaluation of phenotypes will not mislead the search process.

A potential disadvantage of a phenotype representation is that the neighborhood structure may be complicated and designing search operators could be difficult, since the operators need to be versatile and able to avoid infeasible solutions, such as the collision of shapes. We introduce some space manipulation operators in section 5.3 and a generic drop-and-add mutation operator in section 5.4.

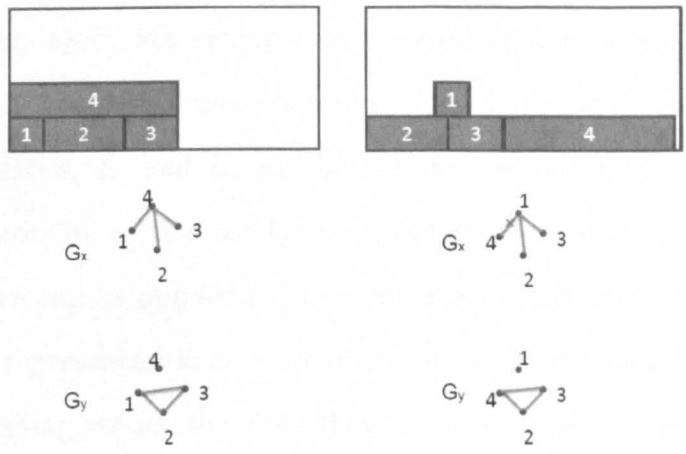
5.2.2 Interval Graph Abstraction

As discussed above, a phenotype representation encompasses comprehensive information regarding a solution. However, it is often necessary, but not trivial, to analyze phenotypic traits, such as comparing similarity between two phenotypes. The real-valued location information of each shape, though intuitive, is complicated to express and compute. We propose a weighted interval graph as an abstraction of the phenotypes, which keeps essential information while also greatly reducing the computational burden.

The weighted interval graphs are based on the packing classes model proposed by Fekete and Sheperd [73] (see chapter 2). In their model shapes are treated as homogeneous nodes with no difference. This model will cause problems as

illustrated by Figure 5.1. Both figures 5.1(a) and 5.1(b) have a valid pair of interval graphs, however if taking item sizes into consideration, in 5.1(b) the edge between nodes 1 and 4 shouldn't exist.

Nevertheless, the interval graphs are still a useful tool to measure similarity between different packings. We extend the model by assigning weights to each node according to each shape's size. Non-weighted interval graphs can be implemented with a binary string data structure, which can be easily compared using a hamming distance. The difference when measuring weighted interval graphs is that, each bit is only a flag indicating if the corresponding weight is to be counted or not, while in Hamming Distance each different bit accounts for one in the distance (i.e. all weights are equal to one). This model can easily be implemented by a $n \times n$ (lower) triangular matrix (n is the number of shapes). Each column (or row) of the matrix is labelled by a shape's index, so the coordinates of each element in the matrix corresponds to the two labelling shapes on the row and column respectively. The weight for each element is calculated by summing up the square of the sizes of the two labelling shapes. If two shapes overlap on a direction the element will be equal to one, or be zero if otherwise.



(a) feasible (b) infeasible

Figure 5.1: Interval graphs ignoring shape sizes can be infeasible

5.3 Phenotype Operators

A search algorithm using phenotype representation, in theory, does not require any decoding heuristics to evaluate a solution, as long as the mutation operators can manipulate phenotypes directly to perform a neighbourhood search. For orthogonal packing problems, one intuitive operator could be defined as shifting a shape to specific coordinates or over a certain distance. Such an operator would, however, be computationally expensive, as it needs to determine if the destination location is feasible and would require complex collision detection. Instead, we use a more generic operator, drop-and-add (DAA, section 5.4) as the search strategy which takes certain shapes out of a phenotype representation (erased from L_p and update L_a) and re-packs them in different places. The DAA mutation strategy suits many different problem domains and is less computationally expensive. However, it requires some space manipulation operators to calculate feasible positions for each shape.

The purpose of the space operators for phenotype representation is to maintain the sets of L_p (for shapes' position) and L_a (for available spaces for future packing). Every time a shape is added or removed from a packing, both L_p and L_a need to be updated. For example, at the beginning of a packing process, L_p is empty and L_a contains all available bins; then shapes are inserted one by one by certain heuristics, L_p and L_a will be modified accordingly. To facilitate the advanced operators, in section 5.3.1, we present two fundamental operators, split and merge, which can be applied to any dimensional orthogonal Euclidean space. The phenotype representation and two operators include comprehensive information about a packing status, therefore they are more general than other methods in the literature, such as those calculating skyline [28] or docking points [146]

The basic operators can also be combined to create more sophisticated operators. We develop three such operators: shift, jostle and relocate (details are shown

in sections 5.3.2 and 5.3.3). These operators are applied on a complete packing, in order to make further improvements without massive structural changes. The shift operator finds gaps and eliminates them by slightly moving neighboring pieces towards a certain direction, while relocate fills gaps with pieces that may be some distance away. The shift and relocate operators may need to be applied recursively to many shapes, since once a shape has been shifted or relocated a new gap may be generated due to the movement. The jostle operator is a variation of the shift operator. The difference is that the jostle operator applies the shift iteratively with changing directions. It starts with shifting pieces in one direction as far as possible. Once no more movements in this direction can be done, it changes the direction to a new direction and checks if any shapes can be shifted in this direction. It mimics the shaking of a container that violently changes directions of all the shapes.

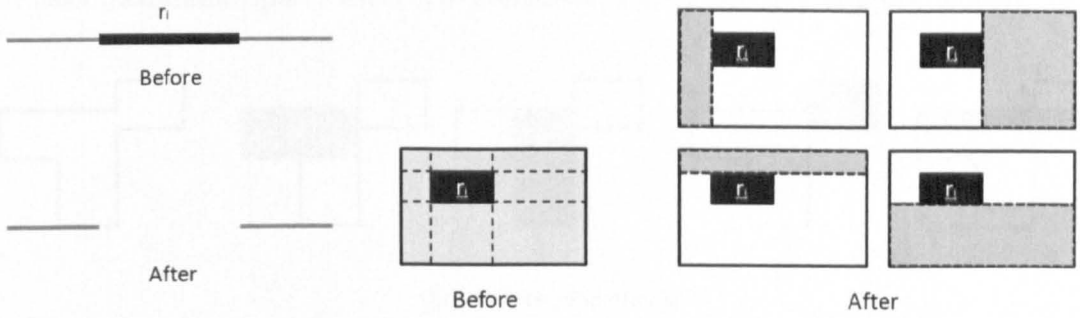
5.3.1 Split and Merge

Split

Space split is a subroutine that is used when adding a shape. It takes three input parameters, a list of prior available spaces $L_a^{'}$, a shape r_i and certain position p_i the shape is to be placed, and it returns a list of posterior available spaces $L_a^{''}$. We will first introduce the basic space split function, followed by explaining how to apply the basic function to a real situation. For multiple shapes, the split routine will be applied to each shape, i.e. pack shapes one by one as if in an online packing scenario.

Figure 5.2 shows the general cases for one and two dimensional space split. It is easy to see that generally the original interval, on each dimension, will be cut in the middle to create two smaller sub-intervals. Therefore the number of newly created subspaces is $2 \times d$ (d is the dimension of the space), e.g. the subspace

generated for one dimensional cases is two, and for two dimensional cases it is four.



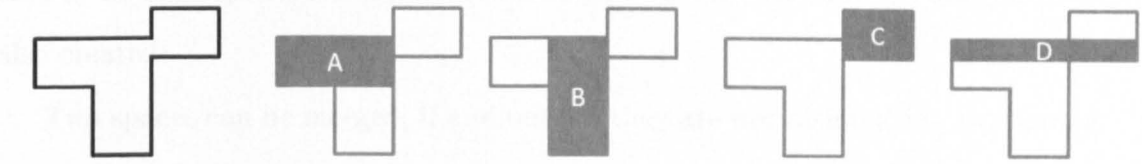
(a) One-dimensional (b) Two-dimensional

Figure 5.2: Changes on available spaces (in grey) before and after placement of r_i (in black)

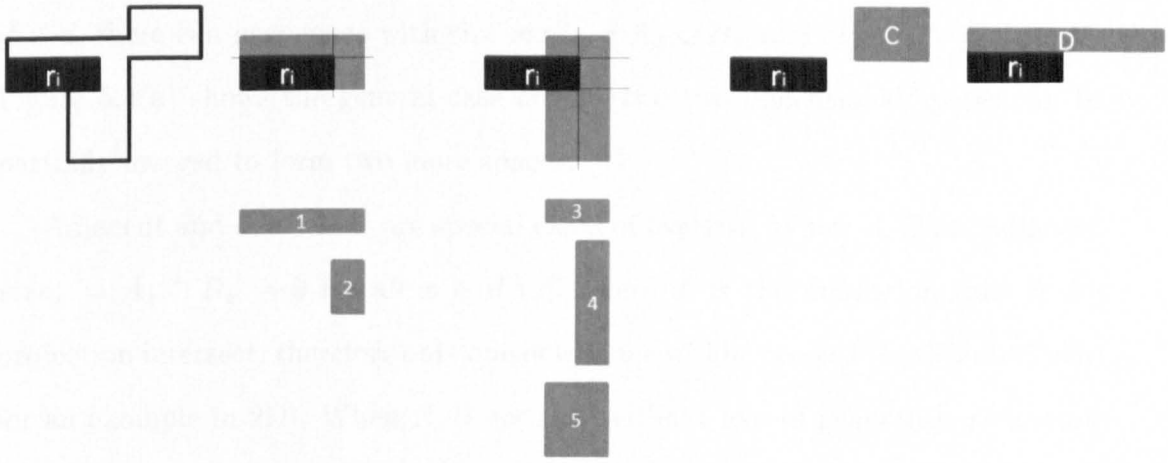
The special cases of the general placement are to align shapes to special positions within a space, e.g. bottom-left corner, as many heuristics usually do. In such cases some of the subspaces have zero value on at least one dimension. Therefore these subspaces can be ignored and omitted from the list of posterior available spaces.

With the basic space split function introduced above, we can tackle the more complicated real situation in the packing process. For instances with two or more dimensions, some available spaces in the list L_a may be overlapping (as in Figure 5.2(b)). Therefore when a shape affects any overlapping areas, it splits several available spaces simultaneously. For each space in L_a, affected by the placement of a shape, we calculate and store the subspaces in a temporary list. In the temporary list some subspaces are enclosed in other spaces, either existing spaces in L_a or other newly created larger subspaces in the temporary list. An example is shown in Figure 5.3. Initially L_a had four available spaces $\{A,B,C,D\}$ (Figure 5.3(a)). After placing r_i, A,B are affected and split into subspaces $\{1,2\}$ and $\{3,4,5\}$ respectively, which are added to the temporary list. Space C is disjoint and D is adjacent to r_i, so they are not affected (Figure 5.3(b)). In the temporary

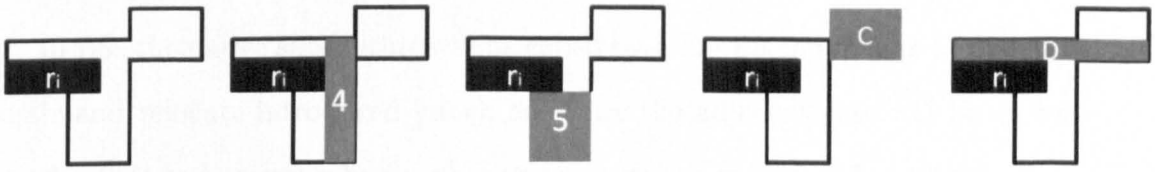
list, subspaces *2,3* are enclosed in *4* and *1* respectively, so they are eliminated. A further check consolidates subspace *1* and *D*, as *1* is enclosed in *D*. Therefore the final list of available spaces after placement of r_i is $\{4,5,C,D\}$ (Figure 5.3(c)).



(a) Before placement



(b) After placement



(c) After consolidating enclosed spaces

Figure 5.3: The list of available spaces is changed from $\{A,B,C,D\}$ to $\{1,2,3,4,5,C,D\}$ after placement of r_i, then to $\{4,5,C,D\}$ after consolidating smaller enclosed spaces.

Merge

The merge subroutine answers the question that given a list of available spaces what are the maximum sizes of a shape that can be accommodated? It is useful

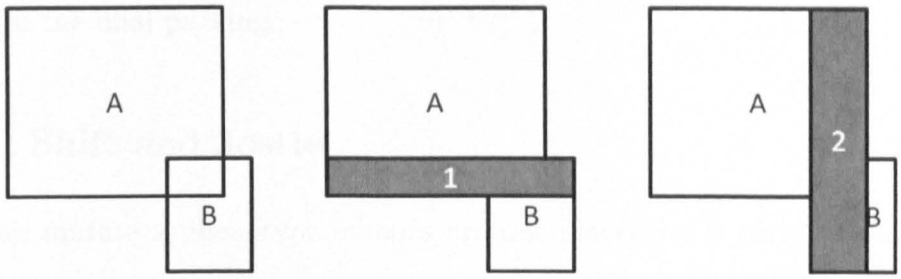
when taking shapes out of a packing. The subroutine takes some lists of spaces as input, and detects if any spaces can be merged to form larger spaces. It is a necessary procedure as in an evolution strategy we need operators that can move shapes in different locations, therefore subspaces can not only be occupied but also created.

Two spaces can be merged, if and only if, they are not disjoint, i.e. overlaping, adjacent or enclosed. Generally, for d dimensional cases, when two spaces A, B overlap, they can merge and create a total of d new spaces. For each dimension $d' \in d$, there is a new space with size $size_{d'} = A_{d'} \cup B_{d'}$ and $size_{x \in d \setminus d'} = A_x \cap B_x$. Figure 5.4(a) shows the general case of how two two-dimensional spaces can be partially merged to form two more spaces.

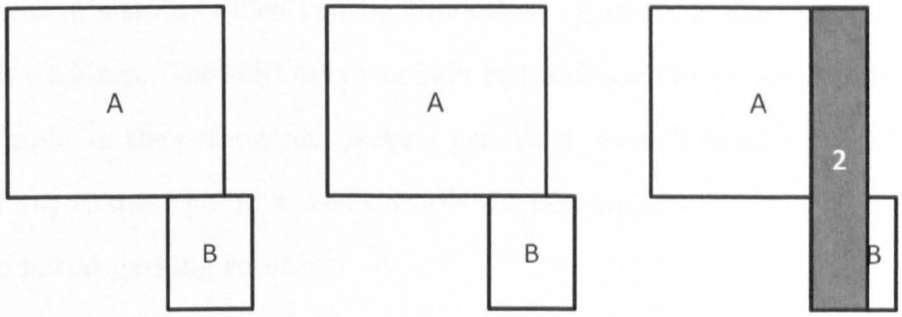
Adjacent and enclosures are special cases of overlap. When A, B are adjacent, $size_x = A_x \cap B_x = \emptyset$ for all $x \in d \setminus d'$ where d' is the dimension that A, B's projection intersect, therefore only one new space will be created (see Figure 5.4(b) for an example in 2D). When A, B enclose (without loss of generality we assume $A \subset B$), $A_{d'} \subset (A_{d'} \cup B_{d'}) \subset B_{d'}$ for all $d' \in d$, i.e. all merged spaces are equal to B, so no space will be added (Figure 5.4(c) shows such an example in 2D).

In ESs the merge subroutine will be called by other phenotype operators (shift, jostle and relocate introduced later), and only the adjacent case will be encountered. This is because when a phenotype operator moves a shape around, and the space previously occupied by the shape will be merged, if and only if, there is any adjacent available spaces. However, a complication in the merge process is that newly merged spaces may be further merged with other spaces. A recursive routine (Algorithm 8) is needed to check if any newly merged space can be further merged with available spaces.

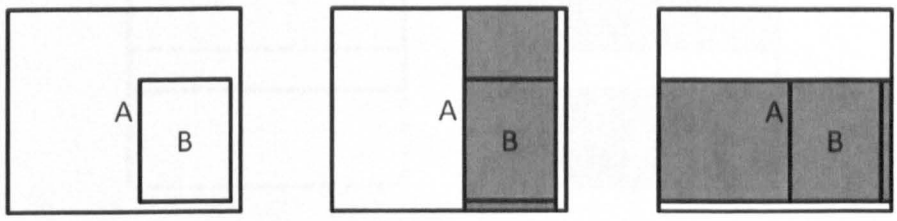
Having introduced the two geometric operators, split and merge for DAA mutation, the next few sections will introduce the three phenotype operators, shift, relocate and jostle, based on the two basic operators, which are optimization op-



(a) Overlap, two new spaces (in grey) created



(b) Adjacent, one new space (in grey) created



(c) Enclose, no new space created (even areas in grey enclosed in A), one existing space B eliminated

Figure 5.4: Merge two spaces A and B

Algorithm 8 Consolidate($Space_{new}, Space_{old}$)

for all $i \in Space_{new}$ **do**
  for all $j \in Space_{old}$ **do**
    if i, j overlap or adjacent **then**
      $Space_{merge} \leftarrow Space_{merge} \cup$ new spaces merged by i, j
    else if i, j enclose or equal **then**
      delete the smaller space
    end if
  end for
end for
$Space_{old} \leftarrow Space_{new} \cup Space_{old}$
if $Space_{merge} \neq \emptyset$ **then**
  Consolidate($Space_{merge}, Space_{old}$)
end if

erators in the final packing.

5.3.2 Shift and Jostle

When we mutate a phenotype, shapes are not placed by a greedy heuristic but at points designated by the mutation strategy. The chances are that gaps may exist between shapes, which can be eliminated by shifting shapes to make more compact packings. The shift operator is in fact a deterministic local improvement. For example, in the orthogonal packing problems, we will check the final packing to see if any shapes can be moved towards the bottom-left corner, which can often improve initial packing results.

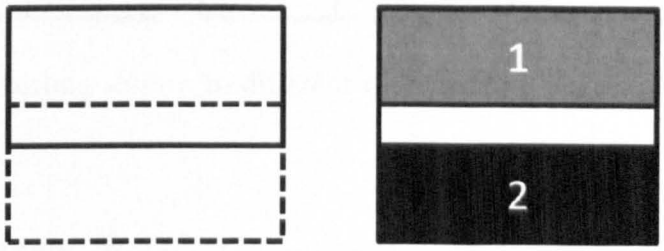


Figure 5.5: Effects of shifting a shape from initial position (solid line) to a new position (dotted line)

The shift operator can be easily implemented with split and merge operators. Figure 5.5 shows how shifting a shape affects the available space list L_a. When a shape shifts from its original position (solid line) to new position (dotted line), the effects are spliting L_a by a dummy shape 2 (in black) and merge a dummy shape 1 (in grey, of the same size as shape 2) with L_a.

Shift operations usually have to be applied recursively. When a shape is shifted, it will return its originally occupied space back to the available space list L_a, which will usually cause a chain reaction as other shapes may be able to shift in the same direction. The shift operation stops when no shapes can be moved in a chosen direction, normally the bottom-left corner.

Jostle was first introduced by Dowsland et al. [57] for irregular shape packing.

We implement a similar idea of jostling that is a variant of shift and mimics the action of shaking a container to reduce the unevenness of the surface. It initially tries to shift all items in one direction, and when no shapes can be shifted further it suddenly changes to a different direction. The motivation behind the operator is illustrated by Figure 5.6. Assuming we initially shift all shapes to a bottom-left position, a better result can be achieved by shifting shape A to bottom-right direction.

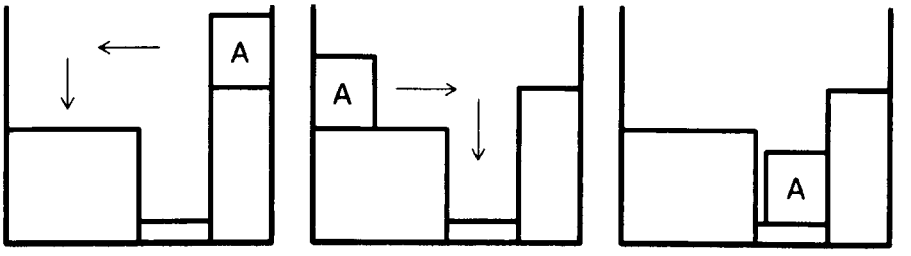


Figure 5.6: Jostling shapes to different directions to achieve better results

5.3.3 Relocate

Unlike shift and jostle operators that move shapes to an adjacent available space, the relocate operator can find a better position which is out of immediate reach of a shape. In Figure 5.7 the operator checks shapes from the top of the packing and gaps from the bottom, to see if a shape can be relocated to a lower position. As with shift operator, the effects of relocating are splitting L_a by the new space (in black) and merge old space (in grey) with L_a. Again, possible chain reactions have to be considered if any shapes can be further shifted or relocated.

5.4 Synopsis of Mutation

Most ESs rely on mutation as the main source of variation to search through phenotype space. Mutation as a neighbourhood search operator defines the ordering of representations on the fitness landscape, and has a great influence on the search

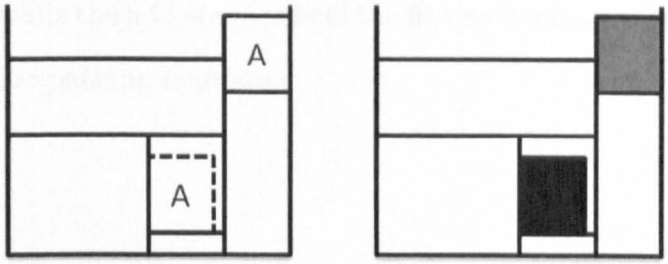


Figure 5.7: Effects of relocating shape A to dotted line position

strategy. By adapting the mutation strength, the step size from an existing solution, mutation operators play a critical role in ES to balance the exploration and exploitation process. To mutate an existing packing, one may be tempted to directly alter the value of each shape's coordinates in L_p. However, this approach can cause shapes to overlap, thus creating infeasible solutions. Repairing such infeasible solutions is often computationally expensive.

In what follows, we briefly explain the principles of a generic two-stage Drop-and-Add (DAA) mutation operator, which repacks certain shapes, similar to the insertion operator in Genetic Algorithms which remove and add an allele in a new position. There are a few variations for both drop and add stages. We will discuss details of the different strategies in section 5.5, but details of the implementation are also elaborated upon in this section as we provide an overview of the DAA mutation operator and, more importantly, the rationale behind the design.

The DAA mutation is comprised of two elementary stages. The drop stage is concerned with how many, and which shapes, are to be removed while the add stage decides how to match the removed shapes with the available spaces. During the drop stage, after each shape has been removed from L_p, the space merge routine will be called to recalculate the available space L_a, similarly at the add stage L_p and L_a will be recalculated iteratively after each shape has been added back. By counting the number and size of shapes being altered, we can quantify the magnitude of the alteration, notated by mutation strength. The drop stage is of the greatest importance with regard to the mutation strength for phenotype

representations, while the add stage affects the fitness landscape by deciding fitness values with specific packing heuristics.

5.4.1 Drop

In chapter 2 we reviewed three general principles (reachability, unbiasedness and scalability) for ESs' mutation strategies in uni-modal and continuous real-value search domain [16]. The DAA mutation operator satisfies the reachability and unbiasedness rules. But the scalability is harder to achieve because of the discrete properties of the packing problems. Considering both quantity and size of shapes provides a way to mitigate the non-differentiable issue. However, with the ES defined by DAA mutation, there is a minimal mutation strength which is alteration of the position of the smallest shape. For adaptation purpose, the number of shapes can be controlled by an integer number from 1 to n exactly, or by a percentage ratio within the range from 0 to 100%. A parametrised probability function calculates which shapes are to be chosen. By tuning the parameter, the probability for each shape can be changed, from preferring small shapes, to a random selection and to preferring large shapes.

5.4.2 Add

Like many other packing heuristics, the insertion of shapes back into a partial packing can be further decomposed into two steps: pre-treatment and matching. Pre-treatment sorts shapes according to non-increasing width or height before packing, which is a deterministic procedure. Unless combined with a stochastic matching strategy, the insertion will be biased and limited to part of the solution space (Chapter 3). On the other hand, placing large shapes earlier often generates good packings, so we use the pre-treatment only to get one of the starting solutions in the initial population, and in subsequent generations we resort to random orders

without sorting.

Matching shapes to available spaces can be done by some standard heuristics such as Best Fit (BF), First Fit(FF). together with Bottom Left or other filling strategies. In theory, a random placement of a shape can be done as long as feasibility is satisfied. We assume the effect of this random matching is equivalent to packing randomly ordered shapes with deterministic heuristics. Empirical results are shown in section 5.7.

5.5 Implementation of Simple ESs

In this section we introduce the implementation of our proposed ES. In particular we focus attention on several variations of the DAA mutation operator and corresponding strategies for adapting the search parameters. The purpose of this section is to examine the dynamics between the phenotype landscape and the mutation strength. It is also necessary to investigate the effects of adapting strategy parameters on such a fitness landscape. Lastly, these basic approaches set the stage for more advanced features and hybrid strategies and provide benchmarks for comparison in later sections.

Algorithm 9 presents a template of a standard ES procedure. From lines 1 to 4, the first generation is initialized randomly. Following the initialization, an iterative loop evolves further generations. Parents are selected from the population in line 8 and produce offspring by mutation in line 10. Before mutation a critical step of ES, shown in line 9, is to adjust the endogenous parameters according to the new results. If we generate multiple children in line 10, a selection among the children will be performed. Such a selection of children is different from the selection of parents in line 8. Although both apply elective pressure for evolution, there is a subtle difference. At line 8, the selection pressure is to filter out unpromising areas of the fitness landscape, while in line 10 the pressure is to specify the intensity of

search around a certain solution. At line 16, the new generation is chosen from the parent and/or offspring population depending on a merge strategy (plus or comma, see section 5.5.3). The evolution process from line 7 to line 16 will be repeated until the stop criteria is met.

Algorithm 9 ES::solve(instance)

1: **for** $i = 1$ to POP_SIZE **do**
2: $s_i \leftarrow RANDOM\_PACK()$;
3: $P^0 \leftarrow INSERT(s_i)$;
4: **end for**
5: generation count $g \leftarrow 0$, initial parameter $\sigma^0 \leftarrow INITIALIZE()$
6: **while** $g++ \leq g_{max}$ **do**
7: **for** $i' = 1$ to NEW_POP_SIZE **do**
8: $s_p \leftarrow SELECT(P^g)$;
9: $\sigma^{g+1} \leftarrow ADJUST(\sigma^g)$
10: $s_c \leftarrow SEARCH(s_p, \sigma^{g+1})$;
11: $P^{g+1} \leftarrow INSERT(s_c)$;
12: **if** $s_c \leq s_{best}$ **then**
13: $s_{best} \leftarrow s_c$;
14: **end if**
15: **end for**
16: MERGE(P^{g+1}, P^g);
17: **end while**
18: return s_{best}.

5.5.1 Endogenous Parameters

Mutation operators introduced in the previous section are the primary source of variation in standard ESs. In this section we present several versions of DAA which implement different strategies with regard to the following questions. How many and which shapes are to be relocated? Which heuristic should be used to replace the piece? These different DAA variants determine how to make a step away from an existing solution. There are two obvious endogenous factors affecting step sizes: the number and size of removed shapes. Correspondingly, we use σ_1 and σ_2 to control the two factors and study their effects separately and jointly.

Another critical factor in tuning mutation strength is the success threshold.

The success rate is defined as the percentage of the descendants better than the parents. In the real-valued continuous domain, the famous 1/5-rule [16] can be easily justified by the discovery of evolution window [16]. However, in the combinatorial search domain, it is uncertain if the rate is still a good choice or even if a constant success rate is reasonable, because the fitness landscape is much more complex and there is a minimum step size requirement. For example, in the bin packing problem, as shapes are not differentiable, the minimum step is equal to the removal of the smallest shape. Therefore we define a third parameter σ_3 to adaptively control the required success threshold. When the mutation reaches the minimum step size and still cannot satisfy a success rate, we will adjust σ_3 to a low level according to certain rules.

For the drop strategy, the following endogenous parameters are investigated:

1. $\boldsymbol{\sigma_1}$ for the number of shapes

   The first endogenous parameter $\sigma_1 \in (0, 100\%]$ controls the percentage of shapes to be repacked, and is the most commonly used parameter for mutation strength in the literature. Though σ_1 is in a real interval, the actual number of shapes is calculated by $\lfloor \sigma_1 \times n \rfloor$. To explore the effects of σ_1, we also adjust it in discrete steps uniformly distributed across the interval rather than continuously. In the experiments in section 5.7, each gap δ between the tested σ_1 is set equal to 10%.

   For adaptive ES introduced in section 5.5.2 the same steps are also used for adjusting σ_1, according to the following rules:

$$\sigma_1 = \begin{cases} \sigma_1 + \delta & \text{if } G_s^1/G^0 > \theta, \sigma_1 < 100\% - \delta \\ \sigma_1 - \delta & \text{if } G_s^1/G^0 < \theta, \sigma_1 > \delta \end{cases}$$

   G^0 denotes parental generation, G_s^1 is the success of offspring in the following generation(s), θ is a required success rate (discussed later in this section).

Note σ_1 is bounded by $0 < \sigma_1 \leq 100\%$.

Like most conventional approaches, using σ_1 alone to control the mutation strength provides only a coarse grained neighbourhood structure. Using only this parameter, there are ambiguities when measuring mutation strength. For example, it may be hard to compare the mutation strength of repack two large shapes against three smaller ones. Therefore, an additional parameter to distinguish shape size is needed.

2. $\boldsymbol{\sigma_2}$ for sizes of shapes

The second endogenous parameter controls the sizes of shapes to be repacked with parameter $\sigma_2 \in \mathbb{R}$ in addition to σ_1. The parameter σ_2 is not a direct representation of the shape sizes, instead it is used in the parametrised function calculating a value P for each shape

$$P(i, \sigma_2) = (\frac{v_i}{v_m})^{\sigma_2}$$

where v_i is the size of shape i and v_m is the median size of all shapes. For large shapes ($v_i > v_m$), increasing σ_2 will increase $P(i)$, and the larger a shape, the faster the P value grows. For small shapes ($v_i < v_m$) the opposite is true, i.e. P increases as σ_2 decreases. So the overall effect is, if tuning σ_2 from high to low the system is more inclined to select small shapes, and vice versa.

With $P(i, \sigma_2)$ for each shape $i \in \{1, 2, \cdots, n\}$, we use roulette wheel selection to chose a shape from those shapes still in the packing. Therefore for a specific shape, j' its probability of being selected is

$$\frac{P(j', \sigma_2)}{\sum_j P(j, \sigma_2)}$$

where $j', j \in J$ are the remaining shapes in a packing.

There are two important properties of the P function. It is monotonic in terms of shape size given a σ_2, that is $\forall v_i \leq v_j$

$$\begin{cases} P(i,\sigma_2) \leq P(j,\sigma_2) & \text{if } \sigma_2 > 0 \\ P(i,\sigma_2) = P(j,\sigma_2) = 1 & \text{if } \sigma_2 = 0 \\ P(i,\sigma_2) \geq P(j,\sigma_2) & \text{if } \sigma_2 < 0 \end{cases}$$

It is also monotonic in terms of σ_2 itself for any two shapes $v_i < v_j$, that is

$$\begin{cases} \frac{P(i,\sigma_2')}{P(j,\sigma_2')} > \frac{P(i,\sigma_2'')}{P(j,\sigma_2'')} & \forall \sigma_2' < \sigma_2'' < 0 \\ \\ \frac{P(i,\sigma_2')}{P(j,\sigma_2')} > \frac{P(i,\sigma_2'')}{P(j,\sigma_2'')} & \forall \sigma_2' > \sigma_2'' > 0 \end{cases}$$

Therefore by adjusting σ_2, we can change the ratio of $\frac{P(i)}{P(j)}$ and affect the probabilities of certain shapes being selected. It is easy to induce the following special cases, for any two shapes i and j remaining in a packing with corresponding sizes of $v_i < v_j$,

$$\frac{P(i,\sigma_2)}{P(j,\sigma_2)} \to -\infty, \text{when } \sigma_2 \to \infty, \qquad \text{largest first} \qquad (5.1a)$$

$$\frac{P(i,\sigma_2)}{P(j,\sigma_2)} = \frac{v_i}{v_j}, \text{when } \sigma_2 = 1, \qquad \text{roulette wheel} \qquad (5.1b)$$

$$\frac{P(i,\sigma_2)}{P(j,\sigma_2)} = 1, \text{when } \sigma_2 = 0, \qquad \text{random} \qquad (5.1c)$$

$$\frac{P(i,\sigma_2)}{P(j,\sigma_2)} = \frac{v_j}{v_i}, \text{when } \sigma_2 = -1, \qquad \text{reciprocal} \qquad (5.1d)$$

$$\frac{P(i,\sigma_2)}{P(j,\sigma_2)} \to \infty, \text{when } \sigma_2 \to -\infty, \qquad \text{smallest first} \qquad (5.1e)$$

The first relationship (5.1a) means that larger shapes have an infinite large probability compared to the smaller ones (equivalent to the largest-first heuristic), while in the last case (5.1e) the smaller ones have infinite prob-

ability (equivalent to the smallest-first heuristic); the second relationship (5.1b) means two shapes are chosen proportional to their sizes (equivalent to roulette wheel selection based on sizes), while the forth proportional to the reciprocal of the sizes; the third relationship (5.1c) means all probabilities are equal (equivalent to a random selection heuristic).

As long as σ_2 is large or small enough (set to between the interval of $[-20, 20]$ for the benchmark instances) the probability function will achieve the desired behaviour of largest (or smallest) first. In the experiment in section 5.7 we also test some settings in between the five special cases in 5.1, i.e. set σ_2 additionally equal to $[2, \frac{1}{2}, -\frac{1}{2}, -2]$ to make finer grained adjustment of σ_2.

Comparing with only σ_1, using both σ_1 and σ_2 to control the mutation strength is in essence refining the mutation strength. With only σ_1 it takes only n steps to reach the maximum mutation strength, i.e. the system samples randomly in the whole search space rather than generating an offspring by inherit traits from a parent solution. With the additional parameter σ_2, if it can take m values, there is a total of $m \times n$ combinations of parameter settings, which offers a finer grained search space. The two parameters can be tuned simultaneously or independently. If tuning them independently, the algorithm will either randomly pick one or arbitrarily assign priority to σ_1 or σ_2. Note the combination of two parameters is still bounded in practice, even though σ_2 is in $\mathbb{R}$, the lower bound is to remove the smallest shape and the upper bound is to remove all shapes.

3. $\boldsymbol{\sigma_3}$ for success rate

$\sigma_3 \in (0, 1)$ prescribes a required success rate for adaptive ESs. The parameter is utilised in the adaptive ES because of two conjectures we have: a constant success rate like the 1/5-success-rule may be too rigid, as it often witnessed in practice, it is harder to generate better offspring as the search

progresses; the combinatorial search domain does not have the asymptotic property that $\sigma \to 0, \quad P \to 1/2$ as in the real domain. In the real domain, asymptotic success rate does not depend on parental status. However, under the phenotype representation and DAA mutation of combinatorial problems, the success rate is clearly parent-dependent. An example is, if the parent solution is a mere stack of all shapes, no offspring solutions will be worse and the success rate trivially equals to one.

The parameter σ_3 is adjusted by a positive constant coefficient α to guarantee its positiveness. We set the coefficient $\alpha = 0.5$, that is

$$\sigma_3^1 = \begin{cases} \sigma_3^0 \cdot \alpha & \text{if } \frac{G_s^1}{G^0} < \sigma_3^0 \\ \\ \frac{\sigma_3^0}{\alpha} & \text{if } \frac{G_s^1}{G^0} > \sigma_3^0 \end{cases}$$

5.5.2 Adaptive ESs

We suspect that the adaptive strategy for the combinatorial domain should be very different from a real-valued domain. In a real-valued domain, due to the existence of asymptotic success rate, i.e. $\sigma \to 0, \quad P \to 1/2$ when current solution is not local (or global) optima, the adaptive strategy can always tune the mutation strength down to maintain a desired success rate and maintain the evolvability. In the real-valued domain, given long enough time, smaller mutation strength can always achieve better (or equal optima) results than larger mutation strengths. In the combinatorial domain, there is obviously a lower bound of mutation strength which is equivalent to repacking the smallest shape, therefore asymptotic mutation strength does not exist. Moreover, it is unlikely for the ESs to keep on improving the solution with the smallest mutation strength, by only repacking the smallest shape. The reason is, when mutation strength is too small, the chances are that other shapes are in a stable position (being highly constrained by each other)

no matter where the smallest shapes are repacked. Therefore a smaller mutation strength does not necessarily outperform a larger mutation strength. For these reasons, we have to adjust the endogenous parameters based on experience and presumptions.

This section extends the simple ES approaches to adaptive strategy by evolving a population consisting of both the strategy parameters and the objective variables (i.e. the shapes and locations). The rationale behind the adaptation is that if a parameter configuration suits a local fitness landscape, it is more likely to produce successful new solutions on average and should be given a better chance of surviving in competition with worse settings.

Algorithm 10 shows an implementation of this idea, which couples each individual with its own parameter configuration. The representation is now a compound of objective variables (the phenotype x) and strategy parameters (σ_1, σ_2). Suppose an individual $(x^{*0}, \sigma_1^{*0}, \sigma_2^{*0})$ is selected to reproduce a child $(x^{*1}, \sigma_1^{*1}, \sigma_2^{*1})$, the phenotype part of the offspring x^{*^1} is mutated from x^{*^0} and controlled by the mutation strength $(\sigma_1^{*0}, \sigma_2^{*0})$. The strategy parameters themselves $(\sigma_1^{*0}, \sigma_2^{*0})$ will also be inherited and updated to $(\sigma_1^{*1}, \sigma_2^{*1})$, depending on whether the offspring x^{*1} is fitter than the parent. Note, in this strategy, the parameter σ_3 is omitted as the success rate is no longer a ratio based on the outcome of evolution of each generation, but a binary indicator of each individual's success. The survival of x^{*1} and $(\sigma_1^{*1}, \sigma_2^{*1})$ are tied together and based on the fitness of x^{*1}.

5.5.3 Exogenous Parameters

Exogenous parameters for ES are mostly parameters controlling the selection pressure, and normally denoted by μ for parental population size, λ for offspring population size, and a selection strategy of either '+' for merge both parent and offspring populations or ',' for selection among only the offspring population.

Selection plays an important role in directing the search to promising areas

Algorithm 10 Self-adaptation on compound individual

1: $Parents \leftarrow$ SELECT($Population$)
2: **for all** individual $i^0 = (x^0, \sigma_1^0, \sigma_2^0) \in Parents$ **do**
3: $x^1 \leftarrow \text{MUTATE}(x^0, \sigma_1^0, \sigma_2^0)$
4: **if** $x^1 < x^0$ **then**
5: $(\sigma_1^1, \sigma_2^1) \leftarrow \text{INCREASE}(\sigma_1^0, \sigma_2^0)$
6: **else**
7: $(\sigma_1^1, \sigma_2^1) \leftarrow \text{DECREASE}(\sigma_1^0, \sigma_2^0)$
8: **end if**
9: new individual $i^1 \leftarrow (x^1, \sigma_1^1, \sigma_2^1)$
10: **end for**

which are estimated on the basis of the fitness of individuals. Setting a proper selection pressure is a key issue in many evolutionary algorithms, including GAs, GPs and ESs. A detailed study on general topics of selection pressure can be found in [54]. We propose to consider the selection pressure from three aspects, which will be examined by empirical study in section 5.7.

1. **Point Selection Pressure** This specifies how many children a parent generates on average, which can be expressed by the ratio of offspring to parents λ/μ. The measure is to quantify the intensity of the search around the local areas of specific points (i.e. the existing solutions). The higher this ratio the more extensively the neighbourhood of existing solutions will be searched. This ratio should be understandably larger than 1 for comma selection, otherwise if the ratio is less than 1 the population size will diminish and there is a risk of population extinction, or if equal to 1 the evolution is mere random walk. Other special cases include, if $\mu = 1, \lambda > 1$ and using plus selection the strategy is equivalent to steepest descent; if both $\mu = \lambda = 1$, the plus selection becomes a simple hill climber.

2. **Population-wise selection pressure** For multi-modal problems, it is important for an algorithm to be able to explore a wide enough search space. One strategy to achieve this goal is to have a sufficient number of parallel agents (other strategies include being able to escape local optimum). For

evolutionary algorithms candidate solutions for reproduction can be deemed as such agents. On the other hand, a good search algorithm has to be able to identify bad solutions to maintain efficiency. In our ESs, this principle is achieved by balancing the ratio $m : \mu : \lambda$ which stands for choosing m candidates from μ parents and λ children. Usually m is chosen to be equal to μ to keep the population size stable.

3. **Generation-wise selection pressure** The ratio $m : \mu : \lambda$ is commonly concerned with only two generations. It is however worth considering choosing candidates from more generations, particularly for the situation where a direct descendant may not show immediate improvement but may be able to do so after a few generations, i.e. some less fit descendants having the potential of getting better results will be discard if only two generations participate in a truncate selection. Therefore we propose a selection strategy of moving multiple-generation window $m : \Sigma\mu : \Sigma\lambda$. Algorithm 11 illustrates this strategy with a generation window of size 3. When merging populations, we use three consecutive generations λ_0, λ_1 and λ_2, rather than just two generations.

Algorithm 11 Multi-generation window

randomly initialise m candidate solutions
repeat
  $\lambda_0 \leftarrow$ REPRODUCE(m)
  $\mu_0 \leftarrow$ POINTSELECT(λ_0)
  $\lambda_1 \leftarrow$ REPRODUCE(μ_0)
  $\mu_1 \leftarrow$ POINTSELECT(λ_0, λ_1)
  $\lambda_2 \leftarrow$ REPRODUCE(μ_1)
  $m \leftarrow$ POPULATIONSELECT($\lambda_0, \lambda_1, \lambda_2$)
until max generation

5.6 Grouping Evolution Strategy (GES)

Unlike a real-valued search domain, for combinatorial optimisation problems, there is not enough theory and empirical evidence to support self-adaptive strategies like those in [16]. Section 5.7.1 will provide some initial insights of the ES behaviour on OPP, which is still far from being comprehensive. For this reason, we resort to experiences in other algorithms to design the adaptive strategy. The GES can be viewed as such an arbitrary adaptive strategy of ES, as opposed to a self-adaptive strategy. In GES, we arbitrarily define two groups for large and small shapes. In its simplest form, GES is indeed a special schema to adapt the mutation strength (section 5.7.2); while in more advanced forms, we hybrid GESs with other algorithms by applying different neighbourhood search methods to the two groups (section 5.7.2).

5.6.1 Definition of Groups

The GES defines groups in a different way from the Grouping Genetic Algorithm (GGA) in chapter 4. It divides the shapes of a heterogeneous instance into only two groups: a critical group for large shapes and a non-critical group for smaller ones. A critical group in the GES is a phenotype representation of a partial solution, i.e. a list of relative coordinates of a proper subset of all shapes $\mathbb{S} \subset \mathbb{R}$, where $s_i \in \mathbb{S}, \iff v_{s_i} > v_{s^*}$. v_{s^*} is a threshold size that specifies the boundary of the two groups. In the real implementation we define the threshold as a percentage. For example, if we define 30% as the threshold for a 20-shape instance, we rank r_i on volume (or height, length or any other measures) such that $r_1 < r_2 < \cdots < r_n$, and the largest 6 ($30\% \times 20$) pieces are members of the critical group.

The threshold can be a fixed ratio or can be adjusted during the search process. Compared to the GGA detecting groups 'on the fly', where there could be an exponentially increasing number of ways to group shapes, the GES has only two

static groups (when the threshold is fixed) or a number of combinations linear to the instance size (when the threshold is dynamic). Due to the different definition, the GES has the following advantages compared to the GGA.

Scalability During a search process, the GGA detects groups 'on-the-fly' and typically has to track an exponentially increasing number of groups. In the GES, however, it tracks only a limited number of critical groups at a time. When instance size increases, the number of combinations of critical groups will only increase linearly. Therefore, the GES avoids the scalability problem of the GGA.

Correlation As explained before, the GES is utilising phenotypes, and has its representations highly correlated to the objective value. The genotype representation on the contrary is indirectly related to the objective value and depends on a heuristic decoder to evaluate its fitness.

Redundancy In simple GAs, redundancy describes the phenomenon that multiple genetic encodings are evaluated to a same solution, which is usually perceived to be undesirable as it makes the search inefficient. The GGA in the previous chapter tries to mitigate the problem to some extent by making the internal structure of a group 'invisible' to its outside neighbouring shapes (treat all groups with same constituent and overall size equal, disregard the internal arrangement). However, redundancy may still arise from the translation from genetic encoding to actual layout, e.g. swapping two shapes (groups) may still generate the same packing. In the GES, redundancy can be easily controlled and effectively eliminated by interval graphs represented packing classes[72] (details are introduced in Section 5.2.2).

5.6.2 Overview of GES

The GES can be regarded as a mixture of some special configurations of the previous standard ESs. In the GES mutation procedure, initially it repacks all shapes in both critical and non-critical groups, which is equivalent to setting $\sigma_1 = 100\%$. The algorithm then proceeds by repacking less shapes in the critical group and all shapes in the non-critical group, as if the σ_1 is gradually reduced from 100% to smaller percentage. Regarding the settings of σ_2, the algorithm repacks all small shapes in the non-critical group which is equivalent to setting σ_2 to a negative value and favour dropping small shapes. It further chooses some bigger shapes in the critical group with a probability selection strategy, which is to change the σ_2 to a mid-high positive value that favours choosing larger shapes.

The GES can even be further developed beyond adapting mutation strength. The key idea of GES is that it can apply different parameter settings or even different search strategies to the critical and non-critical groups. For example, the critical group can adopt Evolution Strategy for neighborhood search while the non-critical group may use hill climbing. The benefit of hybrid GES can be illustrated by the fitness landscape. If we arrange the neighborhood of a solution in such a hierarchical structure that its close neighbors correspond to the mutation of the non-critical group only (which we call an inner circle), and the further neighbours (the outer circle) represent the larger mutation strength that also includes some shapes in the critical group. For example, the nearest neighbor is to mutate only the smallest shape, the next nearest neighbor is to mutate the smallest two shapes, and so on. The critical threshold specifies a boundary, within which is the inner circle and outside which is the outer circle. Intuitively the neighbors within the inner circle repacks small non-critical groups while keeping larger shapes in a similar position, which are likely to be in the same of basins of attraction as the original solution; while the outer circle (mutation to the critical group) are likely to be in different basins of attraction. Therefore, it is natural to adopt different

strategies for the two groups which are in different parts of the landscape. For the critical group, a desirable strategy should be able to explore the landscape and find promising basins of attraction. On the other hand, when searching in the inner circle more exploitive strategies may be more appropriate.

Algorithm 12 illustrates the general framework for the GES which is an iterative search procedure, where s_i is the ith solution, p_c and p_{nc} are critical group and non-critical group of a solution respectively, and $P^g_{(.)}$ represents the gth generation where (.) is the place holder of either non-critical group or critical group. The unique steps of the GES are: during initialisation, the critical group and non-critical group are identified (line 3) and inserted (line 5) into separate populations during evolution. The two groups are also separately selected (line 9), mutated (line 10) and merged into each population (line 17).

Algorithm 12 GES::solve(instance)

1: **for** $i = 1$ to POP_SIZE **do**
2: $s_i \leftarrow RANADOM\_PACK()$;
3: $p_c \leftarrow IDENTIFY\_CRITICAL(s_i)$;
4: $p_{nc} \leftarrow IDENTIFY\_NONCRITICAL(s_i)$;
5: $P^g_c \leftarrow INSERT(p_c), P^g_{nc} \leftarrow INSERT(p_{nc})$;
6: **end for**
7: **while** stop criteria not met **do**
8: **for** $i' = 1$ to NEW_POP_SIZE **do**
9: $p_c \leftarrow SELECT(P_c), p_{nc} \leftarrow SELECT(P_{nc})$;
10: $p_{c'} \leftarrow SEARCH(p_c), p_{nc'} \leftarrow SEARCH(p_{nc'})$;
11: $s_{i'} \leftarrow EVALUATE(p_{c'}, p_{nc'})$;
12: $P^{g+1}_c \leftarrow INSERT(p'_c), P^{g+1}_{nc} \leftarrow INSERT(p_{nc'})$;
13: **if** $result \leq BEST\_RESULT$ **then**
14: $BEST\_RESULT \leftarrow result$;
15: **end if**
16: **end for**
17: PLUS_MERGE(P^{g+1}_c, P^g_c) and MERGE(P^{g+1}_{nc}, P^g_{nc});
18: **end while**
19: return $BEST\_RESULT$.

5.6.3 Implementation of GES

For each of the two groups there are a few search strategies will be tested by empirical study. For the critical groups, the idea is that in the earlier stage of the search the critical group is more likely to mutate and explore wider search spaces, while in the latter stages less pieces will be mutated and the search is controlled to do more exploitive probe around certain promising structures. Beside the general mutation strategies introduced in section 5.5.1, the following special parameter settings of $(\sigma_1, \sigma_2, \sigma_3)$ can be implemented.

1. σ_1 is decreasing from 100% to $\beta \in [0\%, 100\%]$ (the critical ratio, the percentage of critical shapes) according to the search progress ratio of the current iteration count to the maximum iterations allowed, and $\sigma_2 = 0$, i.e. as the search progresses, less number of shapes will be mutated and the selection of shapes is random disregard of their size.

2. similar to the strategy above, but utilizing a roulette wheel selection of shapes based on the size of a shape i^* to the total size of critical shapes $\frac{v_{i^*}}{\sum v_i}$, which assigns smaller mutation probabilities to larger pieces.

For non-critical group, which will always be dropped, we choose from some exploitative mutation strategies:

1. **Random Shuffle** This strategy simply shuffles all shapes before insertion. It is a baseline case to illustrate the properties of close neighbors, and is used for comparison with other strategies.

2. **Genetic Algorithm (GA)** As the non-critical shapes are always removed during the mutation process of GES, the sequence of these shapes being inserted back is used as genetic encoding in a standard GA. A population of the non-critical group is kept independent of and co-evolved with the critical group population.

3. **Hill Climbing (HC)** Unlike the above co-evolution of GA and GES, in the HC strategy each sequence of non-critical shapes is not independent but associated with the phenotype of the critical group. The order is perturbed by swapping the positions of a number of shape-pairs. The new sequence is accepted to replace the old sequence only if the result is better than the original solution. Another greedy strategy, steepest descent, is not tested as the neighborhood size of swapping shapes increases exponentially with the instance size, therefore it is computationally prohibitive to enumerate through all neighbors.

4. **Variable Neighborhood Search (VNS)** When the solution neighborhood can be expressed in a hierarchical structure, such as the fitness landscape defined by DAA mutation, VNS can normally help to explore the neighborhood in a more systematic way. A natural hierarchy for the VNS to classify a set of neighborhood N_k is according to the first parameter σ_1 of the DAA mutation, where $k \in \{1, 2 \ldots, n - n^*)$ (n^* is the number of shapes in the critical groups). The VNS searches iteratively within each layer of the N_k neighbor sets in order.

5.6.4 Fitness Function

Like the Grouping GA (GGA), we consider two types of fitness functions in the GES. The first type evaluates static attributes of groups or final solutions, while the other type takes a more dynamic learning approach by measuring historic performance of candidate groups.

The first type includes fitness functions such as group sizes or final heights of solutions; each of them has its own strengths and weaknesses. Generally, it is desirable to have a fitness function to return evaluations which are highly correlated to values of objective functions. Therefore the final packing result seems to be

a natural choice for the fitness function. However, in many empirical studies it has been found that many alternative fitness functions outperforms the intuitive choice especially when the objective function is deceptive. Therefore, in our experiments we will compare both basic fitness functions, based on group size and final packing height, and investigate various combinations of them by assigning different weights. The critical decision when choosing a static fitness function is to balance deceptiveness and correlation with the objective function.

The second type emphasizes finding promising candidates through an iterative learning process. In the GES, the non-critical shapes can often be packed with the critical shapes in many different ways (thereby generate different packings). If we regard all packings sharing the same critical group as a subset of solution space, a conditional probability can be defined, such as $P(f(h) \leq X|S_i)$ where $f(h)$ is the final height and X is an objective height, i.e. given the group S_i is used to build final packing what is the probability of getting a packing with height less or equal to X. $f(h)$ can be other arbitrary fitness functions, such as average height on historic performance. Indeed, proper choice of the statistic measurement $f(h)$ is critical to differentiate candidate groups, and to guide the search to promising areas. From a statistical aspect, the standard deviation is probably an equally important measure as it reflects the volatility of results when a group is used. There are no clear rules on what fitness function is the best for all cases. Two statistical measures, each having its own merit, will be explored in our experiments: $minimum(h), average(h)$. There can be some other fitness functions made up by various combinations of the elementary functions. In our experiments we will also test the linear combination of average and standard deviation $average(h) - standard\_deviation(h)$.

5.7 Experimental Results

To understand the behavior of the ESs, we carried out extensive empirical studies on the various algorithms designed above. In section 5.7.1, we study the effects of both endogenous and exogenous parameters on the fitness landscape of the standard ESs. In section 5.7.2 we present the results on the Grouping ESs. Table 5.1 below shows all experiments carried out in this chapter.

Section	Description
5.7.1	effects of σ_1 mutation strength
	effects of σ_2 repack heuristic
	effects of σ_3 convergence and success rate
	adjusting σ_1, σ_2 and σ_3
5.7.2	effects of critical ratio
	effects of fitness function
	hybridised strategies

Table 5.1: List of Experiments

The algorithms were implemented in C++ and sun on a grid computer with 2.2GHz CPUs, 2GB memory and GCC compiler. Every experiment has been run 50 times to obtain the average on each metric. Benchmark instances are taken from Burke et al. [28]. In the following sections, we present results on one example instance, complete results on all other instances are very similar and can be found in Appendix B.

5.7.1 Simple ESs

Effects of σ_1 and σ_2

This sub-section presents experimental results that are designed to explore the fitness landscape in relation to the mutation strength, σ_1, σ_2. As explained in section 5.5.1, σ_1 and σ_2 control the number and size of shapes to be dropped in DAA mutation. With regard to the strategy to add shapes back, we use heuristics commonly found in the scientific literature. The first heuristic is to add these

dropped shapes back in a random order. Exogenous parameters are set as (50 + 50), i.e. population size of 50, one parent generates one child with plus merge, and the number of generations exercised is 300.

We first examine separately the effects of σ_1 (each row of the table 5.1(a), in Figure 5.8) and σ_2 (each column of the table 5.1(a), in Figure 5.9). The joint effects of σ_1 and σ_2 are explained later (see Table 5.1(a) and 5.1(b)).

It is easy to note from Figure 5.8, that when σ_1 increases to 100%, all lines converge to the same result highlighted by the dotted rectangle. This means that when $\sigma_1 = 100\%$ (and with random add-back heuristic), the ES is equivalent to random packing regardless of other parameter settings. This effect is also shown in Figure 5.9, the dotted line representing $\sigma_1 = 100\%$ is almost a horizontal line and not sensitive to changes of other parameters.

Perhaps what can be observed with a little more surprise is that many ESs may have worse results than the random packing, as illustrated in both Figure 5.8 and 5.9. This indicates the importance of carefully choosing the mutation strength.

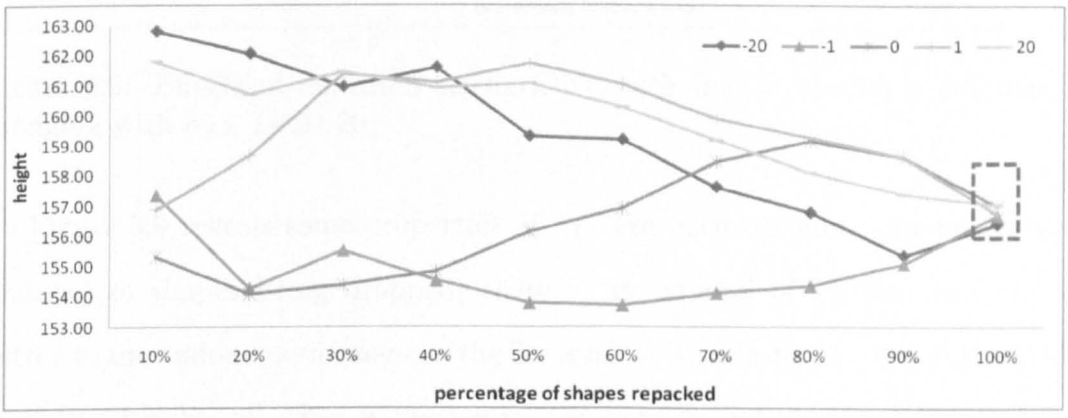


Figure 5.8: Effects of mutation strength σ_2, each line represents a different σ_2 changing with $\sigma_1 \in (0\% - 100\%)$

Especially in Figure 5.8, when $\sigma_2 = 20$ (largest shapes are dropped first), the results are always worse and gradually converge to a random packing when $\sigma_1 \to 100\%$. A similar observation applies, when $\sigma_2 = 1$ i.e. shapes are selected to drop in proportion to their sizes (therefore larger shapes have more chance).

On the contrary, when $\sigma_2 = -1$ (the probability for a shape being dropped is proportional to the reciprocal of its size), a wide choice of σ_1 from 20% to 90% all achieve better results than random packing. This observation confirm our hypothesis that, when we are dealing with heterogeneous instances like those of our benchmark instances, it is better to keep large shapes and mutate smaller ones, which indeed inspired us to treat large and small shapes differently and develop the Grouping ES. Another interesting line in the figure is for $\sigma_2 = 0$ (shapes are chosen randomly to drop with no reference to their sizes), the results initially improve (from $\sigma_1 = 10\%$ to 30%), but get worse when σ_1 increases further.

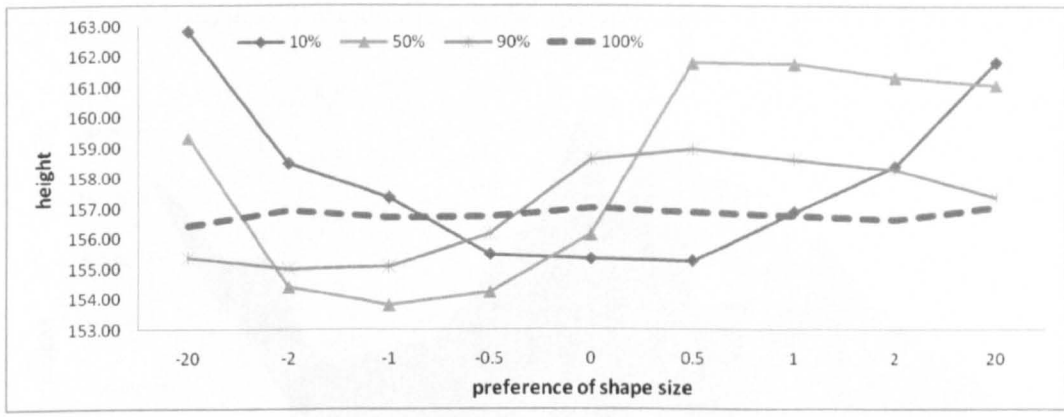


Figure 5.9: Effects of mutation strength σ_1, each line represents a different σ_1 changing with $\sigma_2 \in (-20, 20)$

Figure 5.9 reveals some properties of σ_1. For each different amount (or percentage) of shapes being dropped, there is an interval of σ_2 that can achieve better-than-random performance; the interval will shift to the left side (prefer more smaller shapes) when σ_1 increases. For example, when $\sigma_1 = 10\%$, the interval is $[-1, 1]$; when $\sigma_1 = 50\%$, the interval is $[-2, 0]$; when $\sigma_1 = 90\%$, the interval is $[-0.5, -20]$. For other instances, the exact values of the intervals may vary, but the trend is similar, that is higher σ_1 requires lower σ_2.

The interactive relationship between σ_1 and σ_2 can also be clearly seen from Table 5.2. In 5.2(a) we highlight the 10 best combination of σ_1 and σ_2 out of a total of 90 different settings, which align roughly on the diagonal from top right

to middle-left in the table. If we plot all the values to a 3D chart (as in 5.2(b)), it is even clearer that there exists a valley of mutation strength that can achieve good results.

(a)

n9 / σ2		σ1 10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	162.82	162.08	160.96	161.60	159.32	159.20	157.64	156.80	155.36	156.40
-2		158.48	157.44	155.94	155.90	154.40	154.72	154.10	153.98	155.00	156.92
-1	reciprocal of size	157.36	154.28	155.52	154.52	153.80	153.72	154.12	154.36	155.08	156.70
-0.5		155.48	154.94	154.56	154.08	154.24	154.58	155.28	155.62	156.18	156.72
0	random	155.32	154.10	154.38	154.82	156.14	156.92	158.48	159.18	158.62	157.02
0.5		155.26	155.70	159.02	160.42	161.78	162.22	161.40	159.72	158.96	156.88
1	propotion to size	156.88	158.68	161.40	161.10	161.76	161.04	159.92	159.28	158.62	156.76
2		158.40	161.04	161.84	161.56	161.34	160.54	160.06	159.64	158.28	156.64
20	biggest first	161.82	160.72	161.46	160.94	161.08	160.30	159.20	158.12	157.40	157.04

(b)

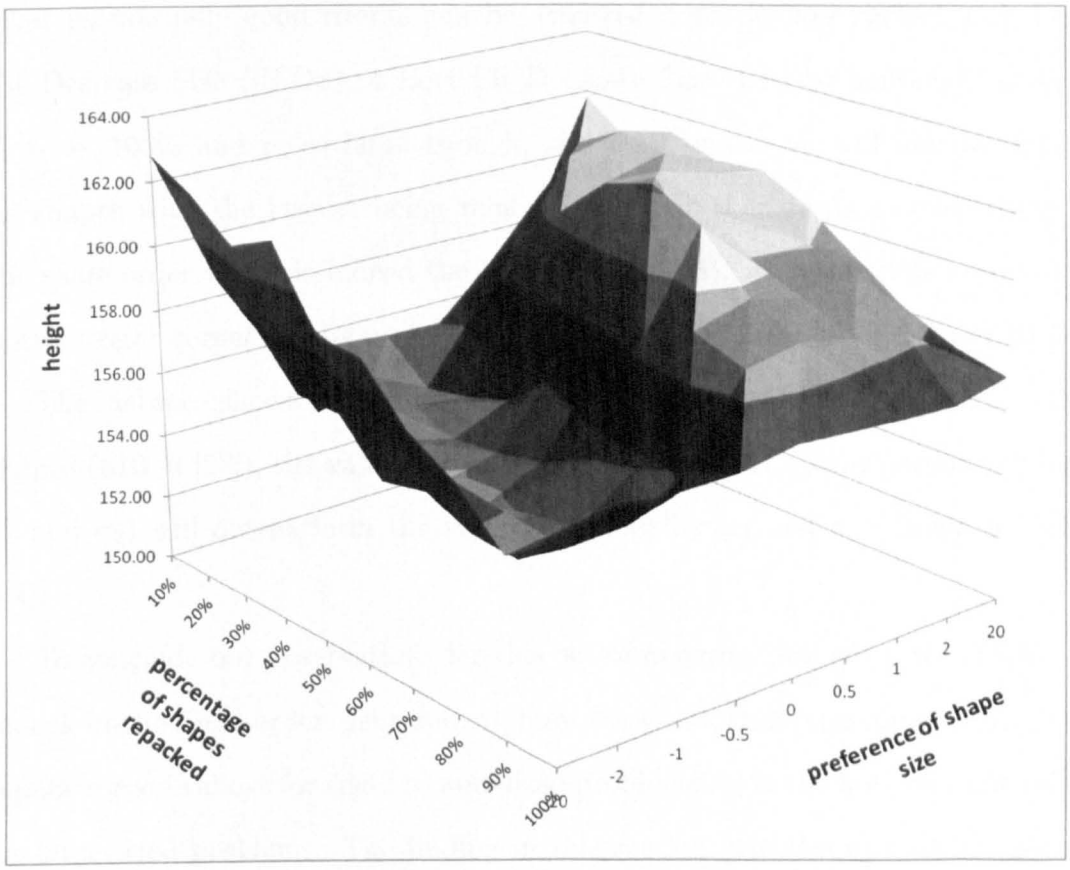


Table 5.2: Effects of mutation strength jointly controlled by σ_1 (parameter for percentage of shapes repacked) and σ_2 (parameter for preference of shape size)

Effects of Repacking Heuristic

The experiments above are using a heuristic that re-packs the shapes in a random order. Another intuitive alternative is to pack them back in the same order that they were dropped. The change will cause an interesting consequence as shown in Table 5.3. We still highlight the 10 best settings of σ_1 and σ_2 in the table 5.3(a). Apart from the good area in the previous experiment, another area around the bottom right corner of the table also has many good results. On the 3D-chart 5.3(b), we can easily see these two disjoint good valleys. This phenomenon resonates with observation of other researchers in the cutting and packing domain. That is, normally good results can be achieved if shapes are packed with First Fit Decrease Size (FFDS) or Best Fit Decrease Size (BFDS) heuristic. In fact, if $\sigma_1 = 100\%$ and σ_2 is large enough, the DAA mutation will always remove all shapes with the largest being removed first. It then re-packs everything in the same order which is indeed the FFDS (or BFDS). Adjacent cells around the bottom right corner can be regarded as a small perturbation of FFDS and BFDS.

The instance shown above has 100 shapes. If the instance size increase to 200 shapes (n10 in [28]), the valley at the bottom right corner (corresponding to high σ_1 and σ_2) will out-perform the valley with smaller σ_1 and σ_2 (shown in Table 5.4).

To conclude our observations for this set of experiments, when we utilise the repack-in-the-same-order heuristic, it may be worthwhile searching in the two separate good valleys for small to mid-sized problems, or in the bottom right valley for large-sized problems. The finding in this section provides us with insight on how to design the self-adaptive ES in section 5.7.1.

Convergence and Success Rate σ_3

In this section we investigate the effects of mutation strength on the success rate and convergence speed. In a real-valued domain, the relationship between muta-

(a)

	n9	σ1									
	σ2	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	161.64	164.12	162.24	162.36	160.80	159.24	159.04	157.60	157.56	162.52
-2		159.56	156.20	158.00	155.92	154.58	154.40	154.26	154.80	156.38	161.84
-1	reciprocal of size	156.00	156.04	154.76	154.82	154.10	154.26	154.28	154.88	156.00	161.70
-0.5		156.28	155.48	154.16	154.04	154.36	154.66	155.32	156.08	157.68	161.36
0	random	154.92	154.08	154.84	155.24	156.32	158.08	158.72	159.64	158.82	156.36
0.5		155.72	155.60	158.20	160.64	160.70	160.70	159.86	157.68	156.16	155.00
1	propotion to size	155.84	157.36	160.68	159.78	159.56	159.32	157.70	156.34	155.06	153.84
2		157.80	156.88	158.40	158.12	157.04	155.94	155.44	154.68	153.92	153.46
20	biggest first	162.04	162.38	160.48	158.44	157.02	155.72	154.16	153.62	153.22	154.24

(b)

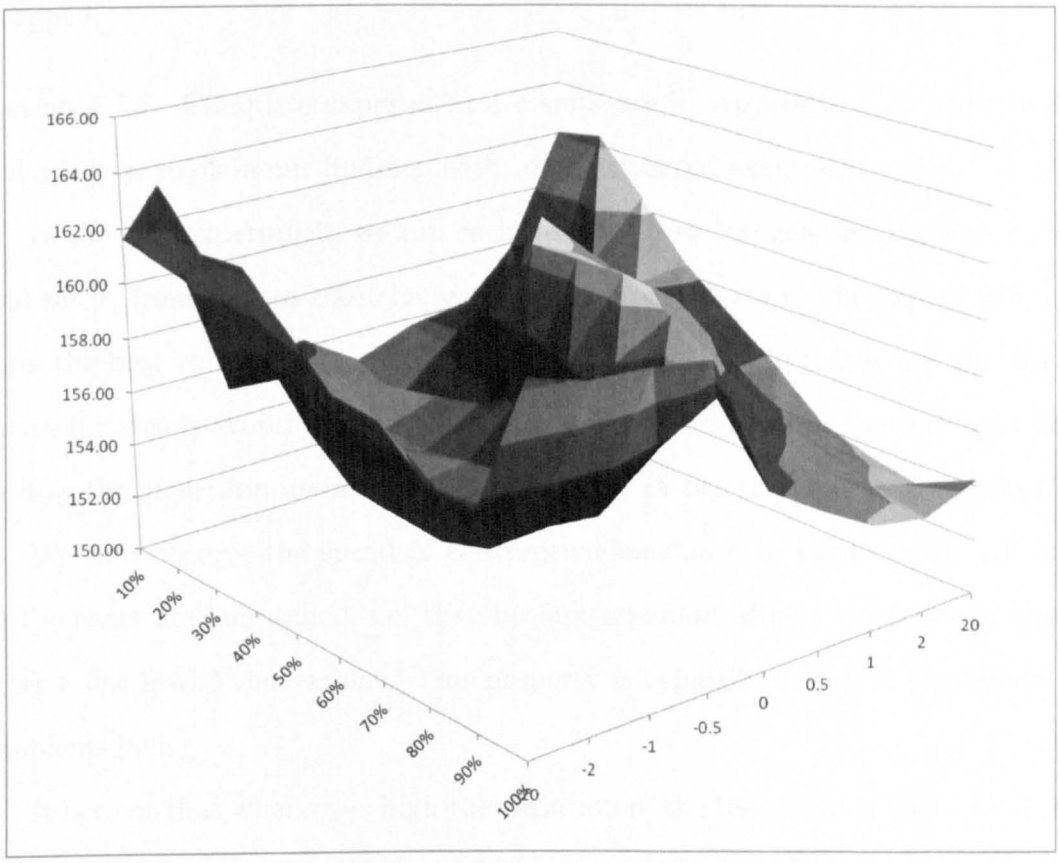


Table 5.3: Effects of mutation strength, shapes are re-packed in the same order as being dropped

tion strength and success rate is the core theory foundation for the self-adaptive strategy [16]. However, some of these theories do not hold for the combinatorial problems like the OPP. As a consequence, we need to consider the self-adaptation strategy differently. In this section we use empirical studies to investigate the dynamics between endogenous parameters and the convergence speed. Based on discussions in this section, we design the self-adaptation strategies in the next

n10		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	158.00	158.46	158.72	158.16	159.40	157.96	157.60	156.34	155.68	157.72
-2		157.88	157.04	157.18	156.50	155.36	154.36	154.04	153.50	154.18	158.10
-1	reciprocal of size	156.90	155.98	155.32	154.90	154.22	153.86	153.70	153.52	154.06	158.26
-0.5		156.36	155.52	155.28	154.56	154.24	154.00	154.04	154.12	154.98	157.98
0	random	155.16	155.14	155.36	156.36	156.54	157.50	157.08	157.18	156.78	154.40
0.5		156.18	156.92	157.86	158.24	157.70	157.74	156.50	155.08	154.16	152.62
1	propotion to size	156.02	157.38	157.32	157.24	156.32	155.28	154.34	153.70	152.86	152.00
2		156.04	156.26	155.92	154.92	154.20	153.30	152.42	152.50	152.22	152.00
20	biggest first	158.42	157.74	156.48	155.08	153.80	153.60	153.02	152.84	152.22	152.00

Table 5.4: A large instance of 200 shapes, re-packed in the same order as being dropped

section 5.7.1. Complete experimental results are in Appendix C.1, while in the following we explain our findings with some extracted examples.

In the first experiment, we run each instance over 300 generations with $\sigma_2 = 0$ and set σ_1 from 10% to 100% incremental at 10% each step. The experiment outputs the best child's fitness in each new generation. (Note this is not the cumulative (or steady-state) best result as in previous sections, since we are interested in how the algorithm maintains the evolvability rather than the final results.)

We first observe the speed of convergence as shown in Figure 5.10. All lines in the chart are long-tailed, i.e. they become stagnant after a quick improvement after a few initial generations. This property is typically found on combinatorial problems [89].

It is clear that when σ_1 is high the variation of the best child is high. As shown in Figure 5.10 the lines for 10% and 30% are smooth while other lines representing higher σ_1 are much more volatile. Consequently, for large σ_1 sometimes the best child may be worse than the best one of previous generation, while small enough σ_1 can normally improve (or not worsen) the current best results.

Lower settings of σ_1, are not only more stable, but can also achieve lower (better) results. As we can see the lines for 10%, 30% and 50% are better than 70%, 90% and 100%. However, among the lower settings, it is not necessarily the case that the lowest setting performs better within the number of generations tested. In fact, for most instances, the best settings is not the lowest one (10%),

but around 30%.

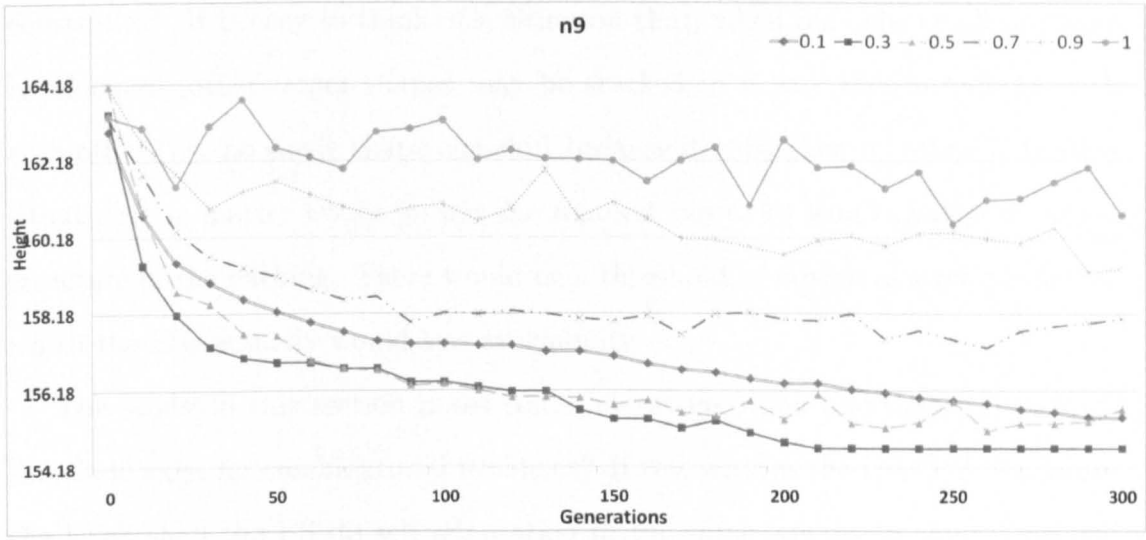


Figure 5.10: Convergence over 300 generations

In an attempt to see, when given long enough time, if the 10% line can eventually take over and close the gap between itself and the 30% line, we run all instances for 30,000 generations. The gap between the two lines are still open (results are in Appendix C.2).

This result suggests an important difference between OPP and real-valued optimization problems. In real-valued search domains, as we discussed in the section on ES theory in Chapter 2 (section 2.6), ESs normally exhibits an asymptote behaviour, i.e. $\sigma_1, \sigma_2 \to 0$ success rate $P \to 1/2$. Therefore, with small mutation strength approximate to 0, the results will definitely be better or equal though it may take a longer time to achieve. Corresponding to this property, in a real-valued domain, when the search is close to optimal, it will have to tune down mutation strength to maintain the evolvability, i.e. the success rate. The difference between discrete and continuous search domain may give us some hints as to the explanation of the observed behavior. Firstly, for combinatorial problems the fitness landscape, mutation of a variable cannot move by an infinitely small step, but there exists a minimal step which is equivalent to replace the smallest shape. Therefore the scalability of a real-valued domain doesn't hold anymore.

Furthermore, in OPP interactions between shapes are more complex and highly constrained. It is easy to think of a situation that, when only the smallest shape is relocated, other larger shapes may be stacked in a way they interlocks with each other (i.e. no single shape can shift because it is blocked by others). In such situations, no matter where we put the smallest piece, we won't change the main structure of the packing. There would be a threshold of mutation strength, below which the ESs actually would loss evolvability.

The study in this section poses some interesting open questions. Does such threshold exist for combinatorial problems? If yes, what is the level? When above the level, shall the ES do self adaptation in the same way as in the real-valued domain? And lastly, what is the expected results in relation to the mutation strength? These are difficult open theoretical questions, but we hope the empirical studies in this chapter can shed some light to assist and motivate other researchers.

Adaptive ES

In light of what we have observed in the previous section, we explore the properties of more dynamic self-adapting ESs in this section. We propose the following strategies based on the previous discussions.

1. Search in the two valleys. When initializing the population, the endogenous parameter will be randomly generated but within the two good valleys identified in section 5.7.1 (orders of shapes are still uniformly random). σ_1, σ_2 are adjusted at the same time but in opposite directions, so as to keep the parameters within the valley.

2. When σ_1 is small and the success rate is too low, in the contrast to a real-valued problem, we increase σ_1 to regain evolvability.

3. Use an adaptive success rate instead of requiring a static success rate of $1/5$ as in the real-valued domain. When the ESs fail to meet the target success

rate, they have a choice of either adjusting σ_1, σ_2 or σ_3. Each time when σ_3 is selected, the algorithms will half the required success rate (typically result in a rate much lower than 1/5).

4. Restart when the population become stagnate. The restart strategy is similar to the one in Chapter 4. The difference is as the ESs operate on phenotypes, they utilise a pair of interval graphs for each packing as an abstraction, and detect if the population is dominated by similar packings from a same packing class.

We compare the results of the dynamic self-adaptive ESs against the ESs with static mutation strength in Table 5.5. Except for the endogenous parameters, other exogenous parameters are set to the same values as the static ESs in Section 5.7.1. The self-adaptive ES is superior on 6 out of 10 instances, and 7 instances when combined with restart mechanism.

Algorithm (σ1 σ2)	Static (10% -20)	Static (20% -20)	Static (30% -2)	Static (40% -1)	Static (50% -.5)	Static (60% 0)	Static (70% .5)	Static (80% 1)	Static (90% 2)	Static (100% 20)	Self_adaptive	Self_adaptive + Restart
n1	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00
n2	54.72	54.32	50.96	50.56	50.00	50.48	51.38	50.40	50.92	54.00	50.00	50.00
n3	55.00	54.56	52.68	51.64	51.72	52.82	53.44	52.76	51.88	53.00	51.59	51.44
n4	91.20	90.44	83.96	82.86	82.68	83.92	84.44	83.70	83.00	87.92	82.68	82.69
n5	108.48	107.96	104.92	105.40	104.94	106.28	107.96	105.88	104.80	107.00	104.63	104.60
n6	106.96	107.80	104.08	102.76	102.80	103.12	104.28	103.52	102.92	103.08	102.64	102.68
n7	118.52	116.72	116.56	108.04	104.62	107.88	109.14	106.24	103.76	114.62	103.66	103.63
n8	89.60	90.38	84.48	83.60	84.26	87.28	87.44	85.66	83.80	84.64	83.61	83.37
n9	162.62	163.00	154.96	154.54	154.48	157.12	159.88	156.32	153.98	154.36	153.85	153.74
n10	158.56	158.78	157.40	155.00	154.26	157.30	156.20	153.70	152.12	152.12	152.15	152.42

Table 5.5: Compare static and self-adaptive ESs (based on maximum computational cost: 30,000 evaluations of fitness function)

5.7.2 Grouping ESs

In this section we examine the results of experiments on the Grouping ES, which either re-pack shapes in a random order or in the sequence that they were removed. These simple GESs can be thought of arbitrary schema that adapts mutation strength from high to low (as if the algorithm is traversing table 5.1(a) from right towards left to pick a combination of σ_1, σ_2). Initially, all shapes in the critical

and non-critical groups are to be mutated, which is equivalent of $\sigma_1 = 100\%$. The re-pack heuristic is slightly different, however, both simply randomly shuffle the orders ($\sigma_2 = 0$) and with non-increasing size ($\sigma_2 = 20$). The re-pack heuristic packs the critical group first, in the order being dropped, prior to packing the non-critical group in a random order. This is equivalent to set σ_2 somewhere in between $[0, 20]$. As the evolution progresses, less critical shapes will be dropped (therefore the total amount of mutated shapes is less), which is in effect decreasing σ_1. Towards the end of the evolution, only non-critical shapes are mutated (no shapes in the critical group will be dropped), σ_1 is now equal to the percentage of non-critical shapes. That is, the GESs will never go beyond this percentage and use σ_1 which is too small. As we can see the simple GES, although an arbitrary adaptation schema, coincidentally fits the findings discussed in the previous section, which suggests that the bottom right settings in Table 5.1(a) is one of the good valleys; and a minimum mutation strength is required to maintain evolvability.

Effects of Critical Ratio

This experiment finds out what is the best critical ratio (percentage of shapes in the critical group). The ratios are defined from 5% to 100% with step size of 5%, other parameters are set as before. We want to see which ratio gives us the best result for each instance. Table 5.6 shows an interesting trend that, when an instance size becomes larger (toward the bottom lines of the table) smaller ratios tend to generate best results. For instance N7 class (around 100 shapes) the best critical ratio is 25% to 30%, while for N6 class (around 75 shapes) the best is 30% to 40% and so on. Complete results for other benchmark datasets are included in Appendix D.

Table 5.6: GES: Best Critical Ratio for Different Sizes of Instances (based on maximum computational cost: 30,000 evaluations of fitness function)

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
N1	210.88	210.98	212.04	212.78	208.57	207.86	208.31	207.93	207.87	208.11	208.52	208.20	208.07	207.50	208.90	207.82	208.72	208.62	207.63	208.41
	210.99	212.46	212.13	208.09	205.50	205.59	205.98	206.22	206.85	205.66	206.85	206.77	207.01	206.85	207.59	207.45	207.38	207.35	206.73	207.67
	210.79	210.65	210.76	210.36	205.41	204.24	203.46	206.93	205.27	205.86	208.33	206.94	208.54	208.42	208.46	209.27	208.29	208.37	208.51	209.12
	210.96	209.79	210.34	205.77	207.38	206.14	205.69	206.18	205.50	207.62	207.26	207.94	208.12	208.19	208.26	207.44	206.68	207.13	206.16	207.27
	208.61	210.36	209.96	209.59	209.31	209.40	210.21	208.86	208.33	208.57	208.28	209.08	208.63	208.68	208.79	209.12	208.86	209.16	208.51	208.65
N2	212.41	212.46	212.81	211.47	211.77	211.92	210.84	210.71	210.76	210.90	210.88	210.87	211.12	211.05	211.23	210.94	211.05	210.83	210.60	210.70
	212.87	212.58	210.63	210.97	210.78	211.42	209.35	209.38	208.58	208.59	208.89	208.86	208.94	208.77	209.12	209.17	209.14	209.20	208.63	208.83
	213.80	212.69	212.62	212.80	213.14	212.73	213.31	212.16	211.42	211.33	211.25	211.68	211.77	211.28	211.76	211.80	212.05	212.01	211.48	211.77
	213.67	213.74	213.37	212.23	213.58	213.36	212.38	210.96	210.48	210.80	210.75	210.87	211.33	211.17	211.35	211.16	211.47	211.60	211.10	211.48
	214.14	214.30	212.37	212.06	211.77	212.43	212.54	212.22	209.85	210.07	210.2	210.33	210.41	210.59	210.64	210.59	210.65	210.47	210.26	210.51
N3	213.56	213.22	212.69	212.90	212.35	211.92	211.52	211.41	211.20	211.15	211.68	211.70	211.91	211.82	211.87	211.73	211.93	212.18	211.95	211.83
	213.98	214.42	213.45	213.10	211.30	211.50	210.84	210.45	210.98	211.17	211.17	211.57	211.76	212.08	212.40	212.12	211.98	212.12	211.65	212.08
	211.22	211.94	211.58	211.24	211.05	211.22	210.10	209.81	209.66	209.76	210.36	210.40	210.55	210.83	210.70	210.49	210.21	210.22	210.06	210.06
	216.07	214.04	213.24	212.76	213.75	212.80	211.85	211.57	211.36	211.44	211.3	211.80	211.66	211.82	211.88	211.65	211.72	211.36	211.03	211.64
	213.09	212.21	211.65	211.10	211.48	211.45	210.95	210.54	210.58	210.48	210.29	210.53	210.72	210.59	210.66	210.61	210.97	211.05	210.62	211.03
N4	214.37	213.61	213.52	212.51	212.54	212.60	211.96	211.48	211.00	211.07	211.51	211.76	212.06	211.76	212.29	212.90	212.55	212.39	212.42	212.67
	215.04	214.68	214.49	214.91	211.61	211.12	210.46	210.66	211.07	211.77	212.31	212.33	212.43	212.67	212.98	213.03	213.03	213.16	212.81	212.61
	212.33	212.35	211.37	211.45	211.53	211.69	211.31	210.83	210.57	210.09	210.17	210.97	211.10	211.29	211.31	211.46	211.45	211.66	211.23	211.49
	212.93	212.54	211.41	210.69	211.10	210.90	210.76	210.46	210.00	210.01	210.87	210.62	210.68	211.78	211.44	211.74	211.78	212.02	211.14	211.38
	213.64	213.21	212.01	212.09	212.23	211.47	209.78	210.16	210.20	210.60	210.8	211.49	211.55	212.33	212.27	212.63	212.22	212.05	212.04	211.86
N5	213.78	213.25	213.22	213.17	213.04	212.78	211.00	210.99	211.92	211.77	212.33	212.76	213.22	213.53	213.65	213.30	213.71	213.52	212.93	213.24
	213.53	212.66	211.98	212.11	211.81	211.68	211.25	210.55	210.68	210.81	211.85	212.04	212.01	212.30	212.56	212.88	212.82	212.44	212.33	212.36
	214.16	213.01	212.64	212.24	211.60	210.67	211.20	211.50	211.93	212.87	212.91	213.19	213.65	213.97	213.99	214.19	214.09	214.78	213.91	213.92
	216.27	216.00	215.70	215.40	215.11	215.41	213.91	213.26	212.80	213.19	213.43	213.38	213.76	213.95	214.59	214.79	215.02	215.58	214.66	214.24
	213.61	213.01	212.47	211.86	212.12	211.98	211.12	211.45	211.78	212.36	212.46	213.05	213.22	213.88	213.37	213.40	213.20	214.67	213.17	213.21
N6	213.29	212.88	212.24	212.13	211.27	210.93	210.23	210.83	211.28	211.68	212.38	212.69	213.55	213.08	213.57	213.59	213.20	214.81	212.80	212.88
	213.13	211.40	211.14	210.44	209.94	209.80	209.94	210.44	211.09	211.81	212.5	213.08	213.22	212.95	213.34	213.37	213.19	215.06	212.90	213.12
	213.05	212.10	211.45	211.28	210.08	209.97	209.47	210.00	210.18	211.45	212.17	212.84	212.95	213.00	213.65	213.57	213.75	214.59	212.72	213.00
	214.60	214.02	213.75	213.68	213.66	212.57	211.66	211.61	212.12	212.46	213.18	213.44	213.59	213.98	214.29	214.42	214.37	215.73	213.99	214.11
	212.11	211.42	210.95	211.38	209.96	209.82	210.37	210.68	211.45	211.76	212.44	212.69	212.96	213.02	213.20	213.27	213.27	214.47	212.90	213.02
N7	212.11	211.31	210.90	209.33	208.46	208.71	209.43	210.57	211.41	211.54	212.47	212.99	213.39	213.55	213.63	213.75	213.47	215.87	213.27	213.01
	211.38	210.59	209.96	209.11	208.70	209.19	210.18	211.04	211.45	212.12	212.6	212.53	213.10	213.61	213.04	213.62	213.82	215.62	213.01	213.17
	211.74	211.30	210.13	209.94	209.07	209.05	209.86	211.16	211.36	212.24	212.79	213.11	213.80	213.52	213.69	213.87	213.97	215.15	212.89	213.14
	213.95	212.88	212.25	210.98	209.91	210.12	210.34	210.98	212.39	212.91	213.41	214.08	214.57	214.77	214.67	214.93	214.92	216.50	214.09	214.31
	211.21	210.41	210.17	210.16	209.60	209.16	209.61	210.01	210.75	211.66	212.32	212.76	213.12	212.88	212.93	213.32	213.35	214.53	212.50	212.87

As we explained before the critical ratio defines the lower bound that σ_1 will traverse, smaller critical ratio actually means higher σ_1. Therefore, it is probably not very surprising to see the results coincide with what we had observed in section 5.7.1, that large-sized problems are more likely to have better results in the bottom right valley in Table 5.2(a).

Effects of Fitness Function

In this section, we evaluate each critical group with different fitness functions. Instead of using only the current packing result to assess the fitness of a critical group, we are interested to see if it is better to evaluate based on its overall historical performance. In this experiment, the algorithm maintains a record for each critical group of all results it participated in a packing. The records are used to calculate certain statistics, in particular, the minimum, the average and the average minus standard deviation. In essence, the experiment is try to find if the algorithm has any learning ability to distinguish good groups from bad ones, therefore guide the search more efficiently.

		Avg			Min			Avg - Std		
inst.	size	avg	min	% of min	avg	min	% of min	avg	min	% of min
n1	10	40.00	40	100.0%	40.00	40	100.0%	40.00	40	100.0%
n2	20	51.02	50	49.0%	50.00	50	100.0%	50.59	50	70.0%
n3	30	52.65	52	34.7%	52.99	52	1.0%	52.90	52	10.0%
n4	40	83.94	83	6.1%	83.76	83	24.0%	83.80	83	19.6%
n5	50	106.05	106	95.0%	106.00	106	100.0%	106.39	105	25.3%
n6	60	103.17	103	83.0%	103.25	103	74.7%	103.35	102	14.0%
n7	70	107.77	106	4.0%	107.36	106	32.0%	106.61	105	40.0%
n8	80	86.06	84	2.0%	86.41	85	1.0%	85.79	85	24.0%
n9	100	157.49	157	50.5%	156.59	155	2.0%	157.68	156	15.2%
n10	200	155.85	154	3.0%	155.49	155	64.6%	156.42	155	1.0%

Table 5.7: GES: Effects of Fitness Function (based on maximum computational cost: 30,000 evaluations of fitness function)

Among the three choices of fitness functions, there is no significant difference among them (see Table 5.7). So we choose the historical minimum as a surrogate fitness (since it is the easiest to calculate compared with the average and the standard deviation), and run another experiment to compare with the simple

GES. The result is shown in Table 5.8.

		Surrogate			Simple GES		
inst.	size	avg	min	% of min	avg	min	% of min
n1	10	40.00	40	100.0%	40.00	40	100.0%
n2	20	50.00	50	100.0%	50.00	50	100.0%
n3	30	50.58	50	43.0%	50.85	50	19.0%
n4	40	81.88	81	13.0%	82.06	81	2.0%
n5	50	103.88	103	19.0%	103.74	103	28.0%
n6	60	102.04	101	3.0%	102.02	101	3.0%
n7	70	103.69	102	4.0%	103.68	102	5.0%
n8	80	83.12	82	7.0%	83.52	82	2.0%
n9	100	153.72	152	1.0%	154.05	153	13.0%
n10	200	154.00	152	1.0%	153.91	153	25.0%

Table 5.8: Compare Surrogate Fitness Function with Simple GES (based on maximum computational cost: 30,000 evaluations of fitness function)

In Table 5.8, the surrogate fitness function does not outperform simple GES significantly, although it wins two more instances. Like the Grouping Genetic Algorithm, it is still an interesting open question whether there is any efficient methods that can identify promising groups.

Different Strategies on Non-critical Group

In the previous experiments, non-critical groups are repacked either randomly or according to the order of them being dropped. The experiments in this section apply more sophisticated search strategies to this group. While the critical groups are evolved by ES, we would like to co-evolve the non-critical group with its own strategy.

Table 5.9 compares the results from previous algorithms (i.e. repack shapes randomly and with non-increasing size) with Genetic Algorithm (GA), Hill Climbing (HC) and Variable Neighbourhood Search (VNS). In these more sophisticated approaches, each individual inherits not only the critical group but also the order of the non-critical group. The GA uses a truncate selection of the top 20 percent to choose another parent and perform crossover on the non-critical group. HC randomly swaps two members of the non-critical group and applies first descent. VNS shuffles more members in the non-critical group with a hierarchical structure

that defines the exchange of one pair as the closest neighbour and all members as the furthest.

		GES + random			GES + BFDS			GES + GA			GES + HC			GES + VNS		
inst.	size	avg	min	% of min	avg	min	% of min	avg	min	% of min	avg	min	% of min	avg	min	% of min
n1	10	40.00	40	100.0%	40.00	40	100.0%	40.00	40	100.0%	40.00	40	100.0%	40.00	40	100.0%
n2	20	50.00	50	100.0%	50.00	50	100.0%	50.00	50	100.0%	50.00	50	100.0%	50.00	50	100.0%
n3	30	50.56	50	44.0%	50.84	50	16.0%	51.02	50	4.0%	50.75	50	25.0%	50.93	50	7.0%
n4	40	81.72	81	28.0%	81.88	81	12.0%	81.60	81	40.0%	81.83	81	17.0%	81.40	81	60.0%
n5	50	103.05	102	2.0%	103.05	103	95.0%	103.02	103	98.0%	103.02	103	98.0%	103.07	103	93.0%
n6	60	102.06	102	94.0%	102.00	102	100.0%	102.15	102	85.0%	102.00	101	11.0%	102.00	102	100.0%
n7	70	102.31	101	24.0%	102.31	102	70.0%	102.93	102	23.0%	102.18	101	11.0%	102.90	101	8.0%
n8	80	82.87	82	19.0%	82.82	82	18.0%	82.68	82	32.0%	82.70	82	30.0%	82.46	82	54.0%
n9	100	152.88	152	16.0%	153.01	152	5.0%	152.86	152	15.0%	152.97	152	3.0%	152.86	152	16.0%
n10	200	152.08	152	92.0%	152.00	152	100.0%	152.00	152	100.0%	152.07	151	1.0%	152.00	151	1.0%

Table 5.9: GES: Different Strategies on Non-critical Group (based on maximum computational cost: 30,000 evaluations of fitness function)

Results in Table 5.9 are hard to differentiate, though VNS and HC do have more winning cases on both average and minimum, and when the results are equal, HC and VNS have slightly higher percentage on hitting the minimum. The reason for these algorithms' performance being equal is possibly due to the reason that on the phenotype fitness landscape of combinatorial problems, the micro-neighbourhood (the close neighbours that can be reached with small mutation strength) around a solution is more random. If the hypothesis is true, it will be hard to optimise on such a micro-neighbourhood with the traditional search algorithms which rely on certain properties of a problem to make it a bit more tractable.

5.8 Summary

Application of Evolution Strategies and phenotype representation on OPP is a less studied area in literature, in spite of great success of these approaches in real-valued (typically continuous, multi-modal) search domains. Some recent theoretical developments have shed light on the application of ES on combinatorial domains such as SAT, ONE-MAX problems [101–103]. These problems are expressed in binary strings and statistical techniques (e.g. finite Markov chain) can be applied to study the algorithms' behaviour. Many other combinatorial optimisation problems, including OPP, are more complicated in phenotype expression,

and direct application of ES is not straightforward. In this chapter, we extend the ES application on OPP and provide empirical evidence to reveal the algorithms' behaviour on OPP.

The first step is to define a phenotype representation for the problem and design operators that can alter the representation to effectively search the neighbourhood. The phenotype is defined by a list, L_p, storing the coordinates of each shape (w.l.o.g. the bottom left corner of each shape). Given an initial empty space of the containers, L_p implies another list of available spaces L_a. Two geometric operators, split and merge, are discussed, which change L_p and L_a whenever we add or remove a shape from a packing. Comparing to other methods in literature, such as calculating the sky-line, and docking points, etc., the representation and the two operators have a great advantage of being general for geometric calculation in orthogonal euclidean space, as the two lists provide full disclosure of information of the status of a packing. They can be used by any orthogonal packing heuristics for any dimension, such as First Fit, and Best Fit.

Based on split and merge operators, we further designed a generic Drop-and-Add (DAA) operator to mutate the phenotype representation. The operator is designed to meet as far as possible the three principles of mutation (reachability, unbiasedness and scalability) and avoid huge computational complexity. Due to the combinatorial nature of OPP, it is difficult to achieve full scalability, however, the DAA provides better control over mutation strength with the help of two parameters, σ_1 for quantity and σ_2 for sizes of shapes.

We discussed the implication of representation and the DAA operator on the fitness landscape and implementation of a basic form of ES. The experiments on the basic ES provides deep understanding on the dynamics between mutation strength and evolution progress. We discovered two good valleys of mutation strength for the OPP problem. We also discussed the impact on the two valleys when the instance size increase and different repacking heuristics are used.

In an attempt to find a good adaptive strategy for the ESs, a set of experiments explored the convergence speed and success rate in relation to mutation strength. The study shows great difference between real-valued and combinatorial domain. On the basis of the findings, we proposed some adaptive strategies which are different from ESs on real-valued problems. The adaptive ESs outperforms ESs with static mutation strength. We also propose GES as another extension of the simple ESs. GESs are more arbitrary adaptive strategies, which also intend to overcome the scalability issue with the GGA in Chapter 4. The GES outperform many other algorithms found in the literature.

There are some interesting open questions. It is still a challenging topic to understand the behaviour of basic ESs on combinatorial search domains, from both a theoretical and an empirical perspective. On the practical side, better design of adaptive strategies is needed, either self-adaptive or arbitrary heuristics. How to overcome the scalability issue and maintain the evolvability? What is asymptotic success rate and the expected hitting time? When search converged around some local optima is there any better strategies to exploit the areas?

Chapter 6

Conclusions

6.1 Summary of Key Contributions

This thesis contributes to the understanding and application of Evolutionary Algorithms on orthogonal packing problems. We first introduce the definition and different models of the problem, followed by overview of the typology of other closely connected problems. The orthogonal packing problems are NP-hard problems and highly constrained by container sizes, other shapes and have different dimensions (for two and more dimensional packing). A survey of the state-of-the-art algorithms, including exact algorithm, heuristics, meta-heuristics and hyper-heuristics is also included, so as to highlight strength and weakness of various approaches.

Chapter 3 contributes to the literature by pointing out the issue that many meta-heuristics' search spaces are restricted to partial solution spaces due to the bias of the low level heuristics they employ, and proposing a hyper-heuristic approach that can mitigate the problem by utilizing multiple heuristic decoders. As an emerging search methodology, theoretical models for understanding the behavior of hyper-heuristics are still needed. We use a framework to explain why the algorithm has the capability to search larger solution space, therefore, raising the level of generality of the search method. The proposed hyper-heuristic utilizes a

heuristic space that can map the same genetic encoding to different phenotypes in the solution space. To test the effectiveness of the algorithm a set of new instances were constructed. Experimental results on the new instances have shown that the hyper-heuristic can find optimal solutions while conventional mate-heuristics fail to do so. When instance size increases, both algorithms cannot solve a problem to optimality, the hyper-heuristic still outperforms meta-heuristics. We also show that the hyper-heuristic has a learning strategy that can evolve the heuristics it chooses. This learning ability enables the algorithm to maintain the same efficiency as conventional meta-heuristics and result in the same convergence speed and comparable results on benchmark instances in the literature.

OPP is an ideal test example for the the Building Block Hypothesis (BBH), one of the fundamental and continuously debated theory of Genetic Algorithms. In Chapter 4 we extend the Grouping Genetic Algorithm (GGA) to higher dimensional OPPs that includes a strip packing (single bin) scenario to examine how BBs (low order schemata) are built up to guide the evolutionary search. We have designed different chromosomes from the original GGA for one-dimensional bin packing. Initially chromosomes are individual shapes as in standard GAs. After packing the initial chromosomes, an additional step is taken to identify blocks (subsets of shapes that can form no-waste meta-rectangles) in each solution. In the following generations, compatible blocks and individual shapes will be mixed together to form variable length chromosomes. We also propose a hierarchical network to describe the block formation process, i.e. to model the build up of BBs from lower order to high order and help us to understand the behavior of GGA.

Three special issues are considered in the implementation of the GGA. Firstly, since the recombination of chromosomes are more complicated, blocks from different parents may be in conflict with others as they may share the same individual shapes. Secondly, evaluation of the blocks to identify promising ones are problematic, because blocks are only partial solutions and evaluation of their fitness

using arbitrary fitness function may be deceptive, i.e. the evaluation of blocks does not reflect the correct paths leading to a successful search of complete solution. Finally, the initial supply of BBs may affect the search results. If the initial population dose not include the correct BBs, receding populations may be directed to wrong areas of the search space. We carried out a set of experiments to test the best settings of these strategies. In particular, we implemented two mechanisms to adaptively rectify the 'wrong' decisions made by preceding populations. The first mechanism is to hybridise the GGA with a tabu list and forbid using a certain number of blocks for some generations if they fail to generate desired results after a certain number of trials. The second mechanism is to restart the whole search process if the search has not made any progress for certain number of evaluations, in case all initial blocks are not good BBs or the search has been stuck at local optima.

As the literature review has pointed out, symbolic encodings like GAs may face the problem of deceptiveness, which means the fitness evaluation on genotypes may miss-align with the phenotypes and guide the search into bad areas. However, direct mutation and recombination of phenotypes of OPP is rarely investigated in the literature due to the computational complexity involved. In Chapter 5, we propose a phenotype representation as well as two operators that unify the calculation of orthogonal packing for any dimensions. The representation is based on the simple data structure of a list of all shapes' bottom left coordinates. Given initial empty space(s), the split operator can calculate incrementally available spaces when each shape is added. Another operator, merge, allows us to calculate the enlarged available spaces when shapes are removed from a packing. The representation and the two operators give us the freedom to easily mutate a packing by a drop-and-add strategy to repack certain shapes, which forms the foundation of developing Evolution Strategies for OPP in the rest of Chapter 5.

Another obstacle for applying ESs to orthogonal packing problems is that

most ESs' adaptive strategies are on the theory basis of real-valued domain. Most importantly the asymptotic success rate $P \to 1/2$ when mutation strength $\sigma \to 0$ and current solution is not local (or global) optima. This behaviour in a real-valued and continuous domain enables the algorithm to always tune down σ to maintain the success rate above a certain required threshold. However, in discrete and combinatorial search domains, such as OPP, the theory is not justifiable. Some recent researches study the ESs behaviour on the combinatorial optimisation problems with simple representation, such as the first hitting time of One-Max problem [103] represented by binary strings. However, the theory is still insufficient to provide comprehensive understanding on the algorithms.

In this thesis the empirical studies of ESs on orthogonal packing problems provide further insights on understanding the ESs behaviour on such a domain, especially, on what are the best settings for mutation strength, what is the relationship between mutation strength and the success rate (convergence speed). The mutation of the simple ES is controlled by two parameters σ_1 for quantity and σ_2 for size of shapes to be repacked. Compared to the conventional mutation strength measurement that counts the number of shapes swapped in genetic encoding, the mutation we designed is a much finer tuning strategy. To find the best static mutation strength, we tested two repack heuristics, one using random orderings and the other using a first-drop-first-add order. We find that for both repack heuristics, there is a same valley of good mutation strength which runs along the diagonal line of the landscape (suggesting combination of large σ_1 with small σ_2 or small σ_1 with large σ_2). The finding suggests a medium-weak mutation strength gives the best results, which coincident with the ES's evolution window found in real-valued domain. When the first-drop-first-add heuristic is applied, another good valley of mutation strength appears towards the corner repacking all shapes with non-increasing sizes. It is also noted, this new valley of good mutation strength outperforms the first valley when instance size increases, which explains why for

many large instances simple greedy heuristics can beat more sophisticated search algorithms, and the best results are achieved by slightly perturbing the greedy heuristic in the second valley.

Another important finding with the simple ESs on OPP suggests the success rate does not necessarily increase when mutation strength decreases, which is quite different from the real-valued domains. The possible explanation is down to the fact that OPP is highly constrained, therefore, the major structure of a packing will not change when too few, and too small shapes, are repacked. This finding also poses a difficulty in the design of the adaptive strategy of ESs, as the conventional 'tuning down' strategy loses its theoretical basis in the combinatorial domain (and is indeed proved inefficient by our empirical study).

To address the adaptation issue of mutation strength for OPP, we proposed two arbitrary adaptive ESs. The first adaptive ES is based on the findings in preceding experiments. It searches in the two good valleys and adjusts σ_1 and σ_2 in opposite directions (one increases the other decreases) at the same time. We also hybridise the strategy with a restart mechanism to prevent the search being stuck at some local optima.

The second adaptive strategy, Grouping Evolution Strategies (GES), decomposes a problem into two groups; critical group for large shapes and non-critical group for small shapes. When mutating a packing, the critical group will stay in place while the non-critical group will be repacked. With further empirical experiments, we found the critical group ratio (percentage of shapes in the critical group) ranges from 20% to 50%. We also found an interesting relationship that larger sized instances require smaller critical ratio. Compared to the GGA in Chapter 4, the GES utilises a less dynamic grouping strategy (only two groups in the GES compared to an increasing number of groups in the GGA). Another potential advantage of GES is that we can apply different search strategies for the two different groups. The empirical results on benchmark instances demonstrate

that the GES is a powerful solver for OPP problems and generates comparable results for some of the best algorithms in the literature.

6.2 Limitations and Suggestions for Future Research

Hyper-heuristics are still a relatively new search paradigm which require more theoretical development. The ultimate objective of hyper-heuristics is to raise the level of generality of search methods. The approach of hyper-heuristics is to use "heuristics to choose heuristics" or "heuristics to generate heuristics", therefore requiring less injection of domain knowledge and human interference. Current mainstream of meta-heuristics need human decisions to choose representations, neighborhood move methods, decoding heuristics (when use symbolic encoding) and evaluation functions. While the proposed hyper-heuristic in Chapter 3 is only one example of how to make intelligent choice among various decoding heuristics, all other aspects of meta-heuristics are subject to incorporation into the general hyper-heuristic paradigm. Correspondingly, other frameworks may be used for modelling and explaining hyper-heuristics search process.

Our hyper-heuristic framework demonstrates one advantage of mapping genetic encodings via the heuristic search space to the solution space, which usually being searched partially with only a single heuristic. Other benefits can be derived by adopting hyper-heuristics' search methodology, such as increasing search efficiency or generating better results. The learning mechanism used in the hyper-heuristics is based on a simple reinforcement learning strategy, which may be further improved by more sophisticated learning strategies such as neural networks, and Bayesian networks, etc. However, the overhead of other learning methods need to be considered, as the current reinforcement learning methodology is very efficient and adds only a slight computational overhead to a standard meta-heuristic. The proposed hyper-heuristic is using 'default' parameters that are commonly found

in the literature. To achieve a fully automated system, adaptive strategies for parameter setting need to be investigated.

In Chapter 4 the hierarchical network model for groups only takes into consideration condensely packed blocks (in the implementation of GGA, we even set the threshold $\theta = 1$ to exclude imperfect groups). In Holland's schemata theorem and the Building Blocks Hypothesis (BBH), low order schemas are not necessarily adjacent. More general models are needed to incorporate these general situations. The network is only an intuitive descriptive model and applicable to only small instances when fully enumerating all potential blocks. For larger instances, it is impractical to enumerate all blocks and the complete Markov Transition Matrix is not available. Therefore, theoretical deriving the probability of hitting certain packing results remains a challenging question, and there is perhaps still a long way to go to ultimately prove (or disprove) the BBH.

Another interesting topic for future work is to see whether there are some ways to distinguish good and bad BBs in the early stages of search. If there are, it would have significant impact on the search strategy. Various learning strategies, such as Bayesian Network, and Neural Network etc., can possibly be hybridised with the GGA in predicting the goodness of BBs.

From an application perspective, the GGA, although a powerful solver for small instances, cannot handle non-guillotine patterns very well. Since for non-guillotine packing, no blocks in an optimal packing have utilisation ratio θ equal to 1 (i.e. groups are always imperfect), the evaluation of blocks are not providing effective guidance for the search process. Lastly, in the GGA identifying blocks is an extra process, which would be computational expensive if not implemented efficiently. In our implementation, we remove bad blocks by setting the threshold θ equal to 1 and remove as early as possible these blocks, so that the algorithm is tracing less potential blocks. It is also interesting to see if any algorithms can be applied to optimise the sub-routine.

Many interesting questions are left for future research in Chapter 5. Firstly, although the two basic orthogonal space operators, split and merge, perform well in practice, understanding their complexity of average and worst cases are still needed. The two operators allow us to mutate a packing with DAA mutation operator, however, recombination of multiple packings is hard to do with the phenotype representation and could be a computationally expensive operation. A possible approach may be like the GGA where compatible blocks are recombined. Again, expensive computation on block identification may be involved.

For the general ES theory on combinatorial search domains like OPP, while our initial investigation shed lights on some phenomena, there are still a lot of fundamental questions to ask, beside the general questions like expected first hitting time for such a complicated representation. Although we discovered the two valleys of good mutation strength setting, we still need to find the reason that cause the two valleys. In particular, it is hard to explain, although maybe intuitive, why the valley representing greedy repacking heuristic will outperform the other valley when instance size increases.

Since there is no asymptotic success rate as in real-valued domain, the convergence properties of ES on OPP is an even more baffling question. Although the arbitrary adaptive strategies generated high quality results for the benchmark instances, without a good understanding on this topic it would be difficult to design better adaptive strategies. Similarly for the GES, what are good evaluation functions of the partial solutions of the critical group? What are good strategies for the different groups, or is the micro-neighbourhood on this phenotype landscape tractable? It is necessary for us to pursue further to understand the problem and the algorithms to fully answer these questions.

Bibliography

[1] N. Alon, Y. Azar, J. Csirik, L. Epstein, S.V. Sevastianov, A.P.A. Vestjens, and G.J. Woeginger. On-line and off-line approximation algorithms for vector covering problems. *Algorithmica*, 21(1):104–118, May 1998.

[2] R. Alvarez-Valdes, F. Parreno, and J.M. Tamarit. A tabu search algorithm for a two-dimensional non-guillotine cutting problem. *European Journal of Operational Research*, (3):1167–1182, December 2007.

[3] R. Alvarez-Valdes, F. Parreno, and J.M. Tamarit. Reactive grasp for the strip-packing problem. *Computers & Operations Research*, 35(4):1065–1083, April 2008.

[4] A.R. Babu and N.R. Babu. Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms. *Int.J.Prod.Res.*, 37(7):1625–1643, May 1999.

[5] A.R. Babu and N.R. Babu. A generic approach for nesting of 2-d parts in 2-d sheets using genetic and heuristic algorithms. *Computer-Aided Design*, 33(12):879–891, October 2001.

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

[7] R. Bai, E.K. Burke, and G. Kendall. Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. *Journal of the Operational Research Society*, 59(10):1387–1397, August 2007.

[8] R. Bai and G. Kendall. A model for fresh produce shelf-space allocation

and inventory management with freshness-condition-dependent demand. *INFORMS Journal on Computing*, 20(1):78–85, January 2008.

[9] B.S. Baker. A new proof for the 1st-fit decreasing bin-packing algorithm. *Journal of Algorithms*, 6(1):49–70, 1985.

[10] B.S. Baker, D.J. Brown, and H.P. Katseff. A 5/4 algorithm for two-dimensional packing. *Journal of Algorithms*, 2(4):348–368, 1981.

[11] B.S. Baker, E.G. Coffman, and R.L. Rivest. Orthogonal packings in 2 dimensions. *Siam Journal on Computing*, 9(4):846–855, 1980.

[12] J.E. Beasley. An exact two-dimensional non-guillotine cutting tree-search procedure. *Operations Research*, 33(1):49–64, 1985.

[13] G. Belov and G. Scheithauer. A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths. *European Journal of Operational Research*, pages 274–294, 2002.

[14] G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171(1):85–106, May 2006.

[15] J.A. Bennell and J.F. Oliveira. The geometry of nesting problems: A tutorial. *European Journal of Operational Research*, 184(2):397–415, January 2008.

[16] H. Beyer and H. Schwefel. Evolution strategies a comprehensive introduction. *Natural Computing*, 1:3–52, 2002. 10.1023/A:1015059928466.

[17] A.K. Bhatia and S.K. Basu. Packing bins using multi-chromosomal genetic representation and better-fit heuristic. *Neural Information Processing*, 3316:181–186, 2004.

[18] A. Bortfeldt. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Journal of Operational Research*, 172(3):814–837, August 2006.

[19] A. Bortfeldt and H. Gehring. A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, 131(1):143–161, May 2001.

[20] J.M. Bourjolly and V. Rebetez. An analysis of lower bound procedures for the bin packing problem. *Computers & Operations Research*, 32(3):395–405, March 2005.

[21] B. Brugger, K.F. Doerner, R.F. Hartl, and M. Reimann. Antpacking - an ant colony optimization approach for the one-dimensional bin packing problem. *Evolutionary Computation in Combinatorial Optimization, Proceedings*, 3004:41–50, 2004.

[22] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J.R. Woodward. Exploring hyper-heuristic methodologies with genetic programming. In J. Kacprzyk, L. C. Jain, C. L. Mumford, and L. C. Jain, editors, *Computational Intelligence*, volume 1 of *Intelligent Systems Reference Library*, pages 177–201. Springer Berlin Heidelberg, 2009.

[23] E.K. Burke, E. Hart, G. Kendall, P Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: an emerging direction in modern research technology. In *Handbook of Metaheuristics*, number 16, pages 457–474. Kluwer Academic Publishers, 2003.

[24] E.K. Burke, R.S.R. Hellier, G. Kendall, and G. Whitwell. Complete and robust no-fit polygon generation for the irregular stock cutting problem. *European Journal of Operational Research*, 179:27–49, 2007.

[25] E.K. Burke, M.R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. *Parallel Problem Solving from Nature - Ppsn Ix, Proceedings*, 4193:860–869, 2006.

[26] E.K. Burke and G. Kendall. Comparison of meta-heuristic algorithms for clustering rectangles. *Comput.Ind.Eng.*, 37(1-2):383–386, October 1999.

[27] E.K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9:451–470, December 2003.

[28] E.K. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52(4):655–671, July 2004.

[29] E.K. Burke, G. Kendall, and G. Whitwell. Simulated annealing enhancement of the best-fit heuristic for the orthogonal stock cutting problem. *Informs Journal on Computing*, 2009.

[30] E. K. Burker, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. A classification of hyper-heuristic approaches. In M. Gendreau and J. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 449–468. Springer US, 2010.

[31] A. Caprara. Properties of some ilp formulations of a class of partitioning problems. *Discrete Applied Mathematics*, 87(1-3):11–23, October 1998.

[32] A. Caprara, A. Lodi, and M. Monaci. Fast approximation schemes for two-stage, two-dimensional bin packing. *Mathematics of Operations Research*, 30(1):150–172, February 2005.

[33] J. Carlier, F. Clautiaux, and A. Moukrim. New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. *Computers & Operations Research*, 34(8):2223–2250, August 2007.

[34] K Chakhlevitch and P. I Cowling. Hyperheuristics: Recent developments. In *Adaptive and Multilevel Metaheuristics*, volume 136, pages 3–29. Springer, 2008.

[35] B. Chazelle. The bottom-left bin-packing heuristic: an efficient implementation. *Ieee Transactions on Computers*, 32(8):697–707, 1983.

[36] B. Chen, A. van Vliet, and G.J. Woeginger. A lower-bound for randomized online scheduling algorithms. *Information Processing Letters*, 51(5):219–222, September 1994.

[37] C.S. Chen, S.M. Lee, and Q.S. Shen. An analytical model for the container loading problem. *European Journal of Operational Research*, 80(1):68–76, January 1995.

[38] P. Chen, G. Kendall, and G.V. Berghe. An ant based hyper-heuristic for the travelling tournament problem. In G. Kendall, editor, *Computational Intelligence in Scheduling, 2007. SCIS '07. IEEE Symposium on*, pages 19–26, 2007.

[39] E.G. Coffman, J. Csirik, D.S. Johnson, and G.J. Woeginger. An introduction to bin packing, 2004.

[40] E.G. Coffman, M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: a survey. In D. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing, Boston, 1996.

[41] E.G. Coffman, M.R. Garey, D.S. Johnson, and R.E. Tarjan. Performance bounds for level-oriented 2-dimensional packing algorithms. *Siam Journal on Computing*, 9(4):808–826, 1980.

[42] P. Cowling and K. Chakhlevitch. Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. *Evolutionary Computation*, 2:1214–1221, May 2004.

[43] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A robust optimisation method applied to nurse scheduling. In Juan Guervós, Panagiotis Adamidis, Hans-Georg Beyer, Hans-Paul Schwefel, and José-Luis Fernández-Villacañas, editors, *Parallel Problem Solving from Nature âĂŤ PPSN VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 851–860. Springer Berlin, 2002.

[44] P. I. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, PATAT '00, pages 176–190, London, UK, 2001. Springer-Verlag.

[45] T. G. Crainic, G. Perboli, M. Pezzuto, and R. Tadei. New bin packing fast lower bounds. *Computers & Operations Research*, 34(11):3439–3457, November 2007.

[46] T.G. Crainic, G. Perboli, M. Pezzuto, and R. Tadei. Computing the asymptotic worst-case of bin packing lower bounds. *European Journal of Operational Research*, 183(3):1295–1303, December 2007.

[47] T.G. Crainic, G. Perboli, and R. Tadei. An interval graph-based tabu search framework for multi-dimensional packing. *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.8933*, 2003.

[48] J. Csirik. On the worst-case performance of the nkf bin-packing heuristic. *Acta Cybern.*, 9(2):89–105, 1989.

[49] J. Csirik, J.B.G. Frenk, and M. Labbe. 2-dimensional rectangle packing - online methods and results. *Discrete Applied Mathematics*, 45(3):197–204, September 1993.

[50] J. Csirik and D.S. Johnson. Bounded space on-line bin packing: Best is better than first. *Algorithmica*, 31(2):115–138, October 2001.

[51] J. Csirik, D.S. Johnson, C. Kenyon, J.B. Orlin, P.W. Shor, and R.R. Weber. On the sum-of-squares algorithm for bin packing. *Journal of the Acm*, 53(1):1–65, January 2006.

[52] J. Csirik, D.S. Johnson, C. Kenyon, P.W. Shor, and R.R. Weber. A self organizing bin packing heuristic. *Algorithm Engineering and Experimentation*, 1619:246–265, 1999.

[53] J. Csirik and G.J. Woeginger. Shelf algorithms for on-line strip packing. *Information Processing Letters*, 63(4):171–175, August 1997.

[54] K. A. De Jong. *Evolutionary Computation: A Unified Approach*. The MIT Press, 1st edition, March 2002.

[55] K.A. Dowsland. Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, 68(3):389–399, August 1993.

[56] K.A. Dowsland and W.B. Dowsland. Packing problems. *European Journal of Operational Research*, 56(1):2–14, January 1992.

[57] K.A. Dowsland, W.B. Dowsland, and J.A. Bennell. Jostling for position: local improvement for irregular cutting patterns. *J.Oper.Res.Soc.*, 49(6):647–658, June 1998.

[58] K.A. Dowsland, E.A. Herbert, G. Kendall, and E. Burke. Using tree search bounds to enhance a genetic algorithm approach to two rectangle packing problems. *European Journal of Operational Research*, 168(2):390–402, January 2006.

[59] H. Dyckhoff. A new linear-programming approach to the cutting stock problem. *Operations Research*, 29(6):1092–1104, 1981.

[60] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, January 1990.

[61] H. Dyckhoff, H.J. Kruse, D. Abel, and T. Gal. Trim loss and related problems. *Omega-Int.J.Manage.Sci.*, 13(1):59–72, 1985.

[62] H. Dyckhoff, Guntram Scheithauer, and J. Terno. Cutting and packing. In M. Dell'Amico, F. maffioli, and S. Martello, editors, *Annotated bibliographies in combinatorial optimization*, pages 393–412. Wiley, 1997.

[63] J. El Hayek, A. Moukrim, and S. Negre. New resolution algorithm and pretreatments for the two-dimensional bin-packing problem. *Computers & Operations Research*, 35(10):3184–3201, October 2008.

[64] L. Epstein and A. Levin. A robust aptas for the classical bin packing problem. *Automata, Languages and Programming, Pt 1*, 4051:214–225, 2006.

[65] L. Faina. An application of simulated annealing to the cutting stock problem. *European Journal of Operational Research*, 114(3):542–556, May 1999.

[66] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2(1):5–30, June 1996.

[67] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. pages 1186–1192, 1992.

[68] O. Faroe, D. Pisinger, and M. Zachariasen. Guided local search for the three-dimensional bin-packing problem. *Informs Journal on Computing*, 15(3):267–283, 2003.

[69] S.P. Fekete. On more-dimensional packing iii: Exact algorithms.

[70] S.P. Fekete. On more-dimensional packing ii: Bounds. 1997.

[71] S.P. Fekete, E. Kohler, and J. Teich. Higher-dimensional packing with order constraints. *Algorithms and Data Structures*, 2125:300–312, 2001.

[72] S.P. Fekete and J. Schepers. A combinatorial characterization of higher-dimensional orthogonal packing. *Mathematics of Operations Research*, 29(2):353–368, May 2004.

[73] S.P. Fekete and J. Schepers. A general framework for bounds for higher-dimensional orthogonal packing problems. *Mathematical Methods of Operations Research*, 60(2):311–329, 2004.

[74] S.P. Fekete, J. Schepers, and J.C. van der Veen. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55(3):569 587, 2007.

[75] H. Foerster and G. Wäscher. Simulated annealing for order spread minimization in sequencing cutting patterns. *European Journal of Operational Research*, 110(2):272–281, October 1998.

[76] A. Freville. The multidimensional 0-1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, May 2004.

[77] G. Galambos and G.J. Woeginger. Repacking helps in bounded space online bin-packing. *Computing*, 49(4):329–338, 1993.

[78] M.R. Garey, R.L. Graham, D.S. Johnson, and A.C.C. Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory Series A*, 21(3):257–298, 1976.

[79] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman, January 1979.

[80] P.C. Gilmore. Cutting stock problem. *Canadian Mathematical Bulletin*, 9(6):737, 1966.

[81] P.C. Gilmore and R.E. Gomory. A linear-programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

[82] P.C. Gilmore and R.E. Gomory. A linear-programming approach to the cutting stock problem .2. *Operations Research*, 11(6):863–888, 1963.

[83] P.C. Gilmore and R.E. Gomory. Multistage cutting stock problems of 2 and more dimensions. *Operations Research*, 13(1):94, 1965.

[84] P.C. Gilmore and R.E. Gomory. Theory and computation of knapsack functions. *Operations Research*, 14(6):1045, 1966.

[85] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[86] D. Goldberg and C. Bridges. An analysis of a reordering operator on a ga-hard problem. *Biological Cybernetics*, 62:397–405, 1990. 10.1007/BF00197646.

[87] D. E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison Wesley, 1989.

[88] D. E. Goldberg, K. Deb, and J. Horn. Massive multimodality, deception, and genetic algorithms. 1992.

[89] C.P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 431–437, 1998.

[90] A. Gomez and D. de la Fuente. Solving the packing and strip-packing problems with genetic algorithms. In *Lecture notes in computer science*, volume 1606, pages 709–718, 1999.

[91] A. Gomez and D. de la Fuente. Resolution of strip-packing problems with genetic algorithms. *J.Oper.Res.Soc.*, 51(11):1289–1295, 2000.

[92] J. F. Goncalves. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 183(3):1212–1229, December 2007.

[93] X.D. Gu, G.L. Chen, and Y.L. Xu. Average-case performance analysis of a 2d strip packing algorithm - nfdh. *Journal of Combinatorial Optimization*, 9(1):14–34, February 2005.

[94] R. Günther. A unified view on hybrid metaheuristics. In F. Almeida, M. Blesa Aguilera, C. Blum, J. Moreno Vega, M. Pérez, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin, 2006.

[95] E. Hadjiconstantinou and N. Christofides. An exact algorithm for general, orthogonal, 2-dimensional knapsack-problems. *European Journal of Operational Research*, 83(1):39–56, May 1995.

[96] R.W. Haessler. A note on computational modifications to the gilmore-gomory cutting stock algorithm. *Operations Research*, 28(4):1001–1005, 1980.

[97] R.W. Haessler. One-dimensional cutting stock problems and solution procedures. *Math.Comput.Model.*, 16(1):1–8, January 1992.

[98] R.W. Haessler and P.E. Sweeney. Cutting stock problems and solution procedures. *European Journal of Operational Research*, 54(2):141–150, September 1991.

[99] P. Hansen and N. Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449 – 467, 2001.

[100] P. Hansen and N. Mladenovic. Variable neighborhood search. In E. K. Burke and G Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 211–238. Springer, 2005.

[101] J. He and X. Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127(1):57–85, March 2001.

[102] J. He and X. Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35, March 2004.

[103] Jun He and Xin Yao. Towards an analytic framework for analysing the computation time of evolutionary algorithms. *Artificial Intelligence*, 145(1-2):59 - 97, 2003.

[104] E.A. Herbert and K.A. Dowsland. A family of genetic algorithms for the pallet loading problem. *Annals of Operations Research*, 63:415–436, 1996.

[105] A.I. Hinxman. The trim-loss and assortment problems a survey. *European Journal of Operational Research*, 5(1):8–18, 1980.

[106] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[107] E. Hopper and B. Turton. Application of genetic algorithms to packing problems - a review. In P.K. Chawdry, R. Roy, and R.K.; Kant, editors, *Proceedings of the 2nd on-line world conferrence on soft computing in engineering design and manufacturing*, pages 279–288. Springer Verlag, London, January 1997.

[108] E. Hopper and B.C.H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *European Journal of Operational Research*, 128(1):34–57, January 2001.

[109] E. Hopper and B.C.H. Turton. A review of the application of meta-heuristic algorithms to 2d strip packing problems. *Artificial Intelligence Review*, 16(4):257–300, December 2001.

[110] I. Ikonen, W. Biles, A. Kumar, and R.K. Ragade. Concept for a genetic algorithm for packing three dimensional objects of complex shape, 1996.

[111] S. Imahori, M. Yagiura, and T. Ibaraki. Local search algorithms for the rectangle packing problem with general spatial costs. *Mathematical Programming*, 97(3):543–569, August 2003.

[112] S. Imahori, M. Yagiura, and T. Ibaraki. Improved local search algorithms for the rectangle packing problem with general spatial costs. *European Journal of Operational Research*, 167(1):48–67, November 2005.

[113] A. Moore J. Boyan. Learning evaluation functions for global optimization and boolean satisfiability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 3–10, 1998.

[114] S. Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88(1):165–181, January 1996.

[115] D. S. Johnson. Near-optimal bin packing algorithms.

[116] D. S. Johnson. Approximation algorithms for combinatorial problems. In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 38–49, New York, NY, USA, 1973. ACM.

[117] D.S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3):272–314, 1974.

[118] D.S. Johnson and K.A. Niemi. On knapsacks, partitions, and a new dynamic-programming technique for trees. *Mathematics of Operations Research*, 8(1):1–14, 1983.

[119] L. V. Kantorovich. Mathematical Methods of Organizing and Planning Production. *Management Science*, 6(4):366–422, 1960.

[120] K. Karabulut and M.M. Inceoglu. A hybrid genetic algorithm for packing in 3d with deepest bottom left with fill method. *Advances in Information Systems, Proceedings*, 3261:441–450, 2004.

[121] C. Kenyon and E. Remila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25(4):645–656, November 2000.

[122] W.S. Khoo, P. Saratchandran, and N. Sundararajan. A genetic approach for two dimensional packing with constraints. *Computational Science - Iccs 2001, Proceedings Pt 2*, 2074:291–299, 2001.

[123] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 146–154, Washington, DC, USA, 2004. IEEE Computer Society.

[124] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[125] J. R. Koza and R. Poli. Genetic programming, 2005.

[126] B. Kroger. Guillotineable bin packing - a genetic approach. *European Journal of Operational Research*, 84(3):645–661, August 1995.

[127] B. Kroger, P. Schwenderling, and O. Vornberger. Parallel genetic packing of rectangles. *Lecture Notes in Computer Science*, 496:160–164, 1991.

[128] B. Kroger and O. Vornberger. Enumerative vs genetic optimization - 2 parallel algorithms for the bin packing problem. *Lecture Notes in Computer Science*, 594:330–362, 1992.

[129] M. Labbe, G. Laporte, and H. Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39(4):616–, July 1991.

[130] K.K. Lai and J.W.M. Chan. Developing a simulated annealing algorithm for the cutting stock problem. *Comput.Ind.Eng.*, 32(1):115–127, January 1997.

[131] C.C. Lee. A simple on-line bin-packing algorithm. *J.ACM*, 32(3):562–572, 1985.

[132] J. Levine and F. Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *J.Oper.Res.Soc.*, 55(7):705–716, July 2004.

[133] J.E. Lewis, R.K. Ragade, A. Kumar, and W.E. Biles. A distributed chromosome genetic algorithm for bin-packing. *Robotics and Computer-Integrated Manufacturing*, 21(4-5):486–495, August 2005.

[134] D. Liu and H. Teng. An improved bl-algorithm for genetic algorithm of the orthogonal packing of rectangles. *European Journal of Operational Research*, 112(2):413–420, January 1999.

[135] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: a survey. *European Journal of Operational Research*, 141(2):241–252, September 2002.

[136] A. Lodi, S. Martello, and D. Vigo. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, 112(1):158–166, January 1999.

[137] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *Informs Journal on Computing*, 11(4):345–357, 1999.

[138] A. Lodi, S. Martello, and D. Vigo. Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research*, 141(2):410–420, September 2002.

[139] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1-3):379–396, November 2002.

[140] A. Lodi, S. Martello, and D. Vigo. Models and bounds for two-dimensional level packing problems. *Journal of Combinatorial Optimization*, 8(3):363–379, September 2004.

[141] A. Lodi, S. Martello, and D. Vigo. Tspack: A unified tabu search code for multi-dimensional bin packing problems. *Annals of Operations Research*, 131(1-4):203–213, October 2004.

[142] W. Mao. Best-k-fit bin packing. *Computing*, 50(3):265–270, 1993.

[143] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *Informs Journal on Computing*, 15(3):310–319, 2003.

[144] S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3):414–424, March 1999.

[145] S. Martello, D. Pisinger, and P. Toth. New trends in exact algorithms for the 0-1 knapsack problem. *European Journal of Operational Research*, 123(2):325–332, June 2000.

[146] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, March 2000.

[147] S. Martello and P. Toth. Algorithm for solution of 0-1 single knapsack problem. *Computing*, 21(1):81–86, 1978.

[148] S. Martello and P. Toth. Solution of the zero-one multiple knapsack-problem. *European Journal of Operational Research*, 4(4):276–283, 1980.

[149] S. Martello and P. Toth. An exact algorithm for large unbounded knapsack-problems. *Operations Research Letters*, 9(1):15–20, January 1990.

[150] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28(1):59–70, July 1990.

[151] S. Martello and P. Toth. Upper bounds and algorithms for hard 0-1 knapsack problems. *Operations Research*, 45(5):768–778, September 1997.

[152] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399, March 1998.

[153] E.A. Mukhacheva and A.S. Mukhacheva. The rectangular packing problem: Local optimum search methods based on block structures. *Automation and Remote Control*, 65(2):248–257, February 2004.

[154] T. Osogami. Approaches to 3d free-form cutting and packing problems and their applications: a survey. Technical report, 1998.

[155] T. Osogami and H. Okano. Local search algorithms for the bin packing problem and their relationships to various construction heuristics. *Journal of Heuristics*, 9(1):29–49, January 2003.

[156] E. Özcan, B. Bilgin, and E. Korkmaz. Hill climbers and mutational heuristics in hyperheuristics. In Thomas Runarsson, Hans-Georg Beyer, Edmund Burke, Juan Merelo-Guervós, L. Whitley, and Xin Yao, editors, *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 202–211. Springer Berlin, 2006.

[157] M. Pelikan, D.E. Goldberg, and E.E. Cantú-paz. Linkage problem, distribution estimation, and bayesian networks. *Evol. Comput.*, 8:311–340, September 2000.

[158] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):758–767, September 1997.

[159] D. Pisinger. A minimal algorithm for the bounded knapsack problem. *Informs Journal on Computing*, 12(1):75–82, 2000.

[160] D. Pisinger. Heuristics for the container loading problem. *European Journal of Operational Research*, 141(2):382–392, September 2002.

[161] D. Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, September 2005.

[162] R. Qu, E. K. Burke, and B. McCollum. Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research*, 198(2):392 – 404, 2009.

[163] P. Ramanan, D.J. Brown, C.C. Lee, and D.T. Lee. Online bin packing in linear time. *Journal of Algorithms*, 10(3):305–326, September 1989.

[164] C. Reeves. Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, 63:371–396, 1996.

[165] P. Ross. Hyper-heuristics. In E.K. Burke and G. Kendall, editors, *Search methodologies: introductory tutorials in optimization and decision support techniques*, number 17, pages 529–556. Springer Science, 2005.

[166] P. Ross, J. G. Marín-Blázquez, S. Schulenburg, and E. Hart. Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics. *Genetic and Evolutionary Computation - Gecco 2003, Pt Ii, Proceedings*, 2724:1295–1306, 2003.

[167] P. Ross, S. Schulenburg, J. G. Marín-Blázquez, and E. Hart. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, volume 6, pages 942–948, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[168] K. Sastry, D. Goldberg, and G. Kendall. Genetic algorithms. In E.K. Burke and G. Kendall, editors, *Search methodologies - introductory tutorials in optimization, search and decision support methodologies*, number 4, pages 97–126. Springer, 2005.

[169] G. Scheithauer and J. Terno. The modified integer round-up property of the one-dimensional cutting stock problem. *European Journal of Operational Research*, 84(3):562–571, August 1995.

[170] G. Scheithauer and J. Terno. Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. *Operations Research Letters*, 20(2):93–100, February 1997.

[171] S.B. Shouraki and G. Haffari. Different local search algorithms in stage for solving bin packing problem. *Eurasia-Ict 2002: Information and Communication Technology, Proceedings*, 2510:102–109, 2002.

[172] A. Soke and Z. Bingul. Hybrid genetic algorithm and simulated annealing for two-dimensional non-guillotine rectangular packing problems. *Engineering Applications of Artificial Intelligence*, 19(5):557–567, August 2006.

[173] A. Steinberg. A strip-packing algorithm with absolute performance bound .2. *Siam Journal on Computing*, 26(2):401–409, April 1997.

[174] P.E. Sweeney and E.R. Paternoster. Cutting and packing problems - a categorized, application-orientated research bibliography. *J.Oper.Res.Soc.*, 43(7):691–706, July 1992.

[175] P. Toth. Dynamic-programming algorithms for the zero-one knapsack-problem. *Computing*, 25(1):29–45, 1980.

[176] J.M. Valério de Carvalho. Lp models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273, September 2002.

[177] A. van Vliet. On the asymptotic worst case behavior of harmonic fit. *Journal of Algorithms*, 20(1):113–136, January 1996.

[178] G. Wäscher, Heike Hauáner, and Holger Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, December 2007.

[179] G.J. Woeginger. Improved space for bounded-space, on-line bin-packing. *Siam Journal on Discrete Mathematics*, 6(4):575–581, 1993.

[180] G.J. Woeginger. There is no asymptotic ptas for two-dimensional vector packing. *Information Processing Letters*, 64(6):293–297, December 1997.

[181] A. Wright, M. Vose, and J. Rowe. Implicit parallelism. In *Genetic and Evolutionary Computation GECCO 2003*, volume 2724 of *Lecture Notes in Computer Science*, pages 211–211. Springer Berlin / Heidelberg, 2003.

[182] A. C. Yao. New algorithms for bin packing. *J.ACM*, 27(2):207–227, 1980.

[183] L.H.W. Yeung and W.K.S. Tang. Strip-packing using hybrid genetic approach. *Engineering Applications of Artificial Intelligence*, 17(2):169–177, March 2004.

Appendix A

New 2D Strip Packing Instances

Item	Instance1	Instance2	Instance3	Instance4	Instance5	Instance6	Instance7	Instance8
1	60:60	13:60	60:13	29:60	60:29	60:24	24:60	27:4
2	60:60	60:60	60:60	23:60	60:23	60:29	29:60	21:7
3	50:50	50:50	50:50	7:50	50:7	50:26	26:50	6:8
4	50:50	50:50	50:50	50:50	50:50	50:4	4:50	26:3
5	40:40	5:40	40:5	9:40	40:9	40:15	15:40	19:13
6	40:40	40:40	40:40	40:40	40:40	40:11	11:40	16:22
7	10:10	7:10	10:7	7:10	10:7	10:7	7:10	7:3
8	10:10	10:10	10:10	10:10	10:10	10:3	3:10	7:7
9	31:30	31:30	31:30	11:30	11:30	12:30	12:30	6:7
10		47:60	60:47	31:60	60:31	60:36	36:60	33:56
11		35:40	40:35	37:60	60:37	60:31	31:60	39:53
12		3:10	10:3	43:50	50:43	50:24	24:50	44:42
13				31:40	40:31	50:46	46:50	24:47
14				3:10	10:3	40:25	25:40	21:27
15				20:30	20:30	40:29	29:40	24:18
16						10:3	3:10	3:7
17						10:7	7:10	3:3
18						19:30	19:30	25:23
19								27:56
20								21:53
21								6:42
22								26:47
23								19:27
24								16:18
25								7:7
26								7:3
27								6:23
28								33:4
29								39:7
30								44:8
31								24:3
32								21:13
33								24:22
34								3:3
35								3:7
36								25:7
	Strip width: 151							

Appendix B

Effects of Mutation Strength of ES

Average of Minimum
(first drop first add, different location not required)

σ1: percentage of shapes being dropped;
σ2: parameter of size pereference, used in formula (Vi / Vm) ^ σ2 where Vi is the volume of a shape, Vm is the median volume;
Each value is an average over 50 runs;
Highlighted values are the 10 best results for each instance, i.e. the best settings of σ1 and σ2 for each instance.

n2		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	54.92	53.40	54.36	53.52	53.52	53.32	54.12	53.48	53.20	54.60
-2		53.88	51.68	51.52	51.28	50.48	50.00	50.00	50.80	52.80	54.00
-1	reciprocal of size	53.16	51.44	50.32	50.24	50.72	50.00	50.00	50.48	51.56	53.32
-0.5		54.02	52.20	50.56	50.56	50.00	50.00	50.00	50.92	52.00	52.06
0	random	53.36	51.80	51.32	50.00	50.60	50.72	50.24	52.04	51.88	51.52
0.5		53.70	51.64	51.28	51.76	50.56	50.24	51.28	51.84	52.12	50.48
1	propotion to size	53.70	52.70	51.92	51.76	50.60	50.72	51.48	51.76	51.04	50.00
2		54.20	53.40	52.72	51.64	51.28	50.00	50.00	50.00	50.12	50.36
20	biggest first	55.00	55.50	54.64	54.56	54.80	53.36	54.00	53.08	53.40	54.00

n3		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	55.12	54.76	54.24	53.72	53.28	53.48	53.56	53.46	54.30	55.72
-2		54.08	52.84	52.60	52.12	51.68	51.52	52.00	52.24	53.12	55.00
-1	reciprocal of size	53.48	52.40	52.40	51.64	51.58	51.76	51.88	52.88	53.24	54.20
-0.5		52.82	52.38	51.84	51.32	51.60	51.76	52.12	52.92	53.04	54.04
0	random	53.20	52.00	51.66	51.70	51.82	52.76	53.12	53.64	53.54	52.86
0.5		53.00	52.12	52.68	52.12	53.00	53.76	53.68	53.34	52.80	52.00
1	propotion to size	53.28	52.92	52.36	52.40	53.76	53.80	53.48	52.76	52.20	51.64
2		54.48	52.76	52.48	53.24	53.40	52.92	52.52	52.00	52.00	51.88
20	biggest first	55.80	55.08	54.86	54.30	54.60	53.80	53.24	52.48	52.24	52.76

n4		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	89.64	88.60	88.44	90.36	87.88	85.04	85.60	87.00	85.58	90.52
-2		86.08	84.72	84.92	84.24	83.48	82.76	83.00	84.00	84.12	87.80
-1	reciprocal of size	86.16	83.76	83.16	82.96	82.28	82.88	83.00	83.52	84.00	87.12
-0.5		84.24	83.52	83.08	83.08	82.76	83.24	83.28	83.84	83.92	86.68
0	random	83.92	83.44	82.80	82.86	83.28	83.92	84.12	84.08	84.32	84.00
0.5		83.90	83.44	83.56	83.56	84.08	84.32	84.36	84.12	84.00	83.40
1	propotion to size	84.56	83.88	83.76	84.36	84.34	84.18	83.90	83.88	83.76	82.74
2		85.24	84.60	83.88	83.96	83.88	83.78	83.48	83.06	83.08	82.24
20	biggest first	91.60	90.32	88.12	88.20	86.64	86.16	85.44	84.68	84.60	87.96

n5		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	108.56	107.72	107.80	106.64	106.76	106.56	106.64	106.40	108.28	109.08
-2		107.00	106.24	105.04	106.00	105.00	104.80	105.36	105.92	107.64	108.68
-1	reciprocal of size	106.24	105.64	105.56	104.72	105.32	105.52	105.36	105.88	107.20	108.76
-0.5		106.08	105.16	104.84	105.04	105.52	105.68	106.08	106.12	106.36	107.52
0	random	105.80	105.28	105.28	105.40	106.12	106.62	106.92	107.24	106.52	106.24
0.5		105.96	105.60	105.88	107.04	107.28	107.08	107.16	106.50	106.24	105.48
1	propotion to size	106.36	105.56	106.66	106.92	108.04	107.48	106.64	106.00	105.12	104.84
2		106.80	106.08	106.20	106.92	106.54	106.44	105.56	105.20	104.88	104.60
20	biggest first	109.12	107.84	108.24	107.84	107.68	106.48	105.76	105.88	105.32	107.00

Average of Minimum
(first drop first add, different location not required)

n6		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	107.40	106.76	106.88	106.64	106.56	105.68	104.52	103.88	104.88	106.84
-2		105.40	104.48	103.28	103.56	102.68	103.24	102.88	103.16	103.48	106.32
-1	reciprocal of size	104.44	103.16	102.84	103.16	102.52	102.82	102.88	102.88	103.60	106.04
-0.5		103.78	103.24	102.44	103.06	102.58	102.80	102.92	103.04	103.60	104.88
0	random	103.54	102.72	102.88	102.52	102.96	103.00	103.28	103.72	104.00	103.72
0.5		103.64	102.56	102.56	103.36	104.16	104.24	104.48	104.08	103.64	103.00
1	propotion to size	104.32	103.24	103.56	104.20	104.88	104.64	103.76	103.64	103.00	102.72
2		105.02	104.08	104.04	104.04	104.24	103.96	103.40	103.00	102.76	102.00
20	biggest first	106.96	106.76	107.44	106.56	105.80	104.44	104.46	103.48	103.12	103.12

n7		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	119.08	117.84	117.00	118.60	116.88	114.88	113.72	113.32	112.94	117.24
-2		116.40	114.96	114.84	113.82	109.42	109.28	105.00	104.12	106.80	115.16
-1	reciprocal of size	114.80	111.96	108.48	107.32	105.52	103.64	105.12	104.54	107.16	114.08
-0.5		109.74	108.50	106.00	103.92	105.04	104.26	104.08	104.66	107.56	111.24
0	random	107.76	107.08	105.20	104.76	105.32	107.64	109.16	110.38	109.82	109.22
0.5		106.60	105.96	108.80	112.48	110.68	109.96	109.08	108.44	107.28	105.56
1	propotion to size	107.56	109.20	111.00	110.48	108.98	107.30	107.00	106.76	105.52	103.88
2		109.12	109.40	108.98	107.76	105.64	105.24	105.24	104.56	103.48	102.04
20	biggest first	115.40	115.04	114.40	111.60	110.64	109.16	108.52	107.80	107.64	115.40

n8		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	90.44	90.12	89.76	89.20	87.64	86.00	85.28	85.12	86.64	91.44
-2		87.20	87.00	84.16	84.00	83.76	83.76	84.20	84.94	86.14	90.48
-1	reciprocal of size	86.20	84.20	83.76	83.64	83.60	84.14	84.44	84.72	85.90	89.46
-0.5		84.80	84.20	83.96	84.12	84.28	84.56	85.12	85.20	86.48	88.40
0	random	84.72	84.36	84.64	85.50	86.72	87.12	87.56	87.04	86.80	85.48
0.5		84.60	85.08	87.50	88.64	89.04	88.66	87.24	86.66	84.96	84.20
1	propotion to size	85.12	86.78	88.86	88.34	88.28	87.56	86.24	85.80	84.72	83.78
2		86.64	86.78	87.88	87.74	87.46	86.24	85.60	84.76	83.84	83.76
20	biggest first	90.40	90.44	88.76	87.76	86.66	85.54	84.58	84.54	83.94	84.80

n9		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	161.64	164.12	162.24	162.36	160.80	159.24	159.04	157.60	157.56	162.52
-2		159.56	156.20	158.00	155.92	154.58	154.40	154.26	154.80	156.38	161.84
-1	reciprocal of size	156.00	156.04	154.76	154.82	154.10	154.26	154.28	154.88	156.00	161.70
-0.5		156.28	155.48	154.16	154.04	154.36	154.66	155.32	156.08	157.68	161.36
0	random	154.92	154.08	154.84	155.24	156.32	158.08	158.72	159.64	158.82	156.36
0.5		155.72	155.60	158.20	160.64	160.70	160.70	159.86	157.68	156.16	155.00
1	propotion to size	155.84	157.36	160.68	159.78	159.56	159.32	157.70	156.34	155.06	153.84
2		157.80	156.88	158.40	158.12	157.04	155.94	155.44	154.68	153.92	153.46
20	biggest first	162.04	162.38	160.48	158.44	157.02	155.72	154.16	153.62	153.22	154.24

n10		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	158.00	158.46	158.72	158.16	159.40	157.96	157.60	156.34	155.68	157.72
-2		157.88	157.04	157.18	156.50	155.36	154.36	154.04	153.50	154.18	158.10
-1	reciprocal of size	156.90	155.98	155.32	154.90	154.22	153.86	153.70	153.52	154.06	158.26
-0.5		156.36	155.52	155.28	154.56	154.24	154.00	154.04	154.12	154.98	157.98
0	random	155.16	155.14	155.36	156.36	156.54	157.50	157.08	157.18	156.78	154.40
0.5		156.18	156.92	157.86	158.24	157.70	157.74	156.50	155.08	154.16	152.62
1	propotion to size	156.02	157.38	157.32	157.24	156.32	155.28	154.34	153.70	152.86	152.00
2		156.04	156.26	155.92	154.92	154.20	153.30	152.42	152.50	152.22	152.00
20	biggest first	158.42	157.74	156.48	155.08	153.80	153.60	153.02	152.84	152.22	152.00

Average of Minimum
(first drop first add, different location)

σ1: percentage of shapes being dropped;
σ2: parameter of size pereference, used in formula (Vi / Vm) ^ σ2 where Vi is the volume of a shape, Vm is the median volume;
Each value is an average over 50 runs;
Highlighted values are the 10 best results for each instance, i.e. the best settings of σ1 and σ2 for each instance.

n2		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	54.72	54.32	54.16	53.44	53.88	53.96	53.96	53.48	53.92	54.20
-2		53.72	51.26	50.96	50.24	50.24	50.00	50.00	50.00	52.76	54.12
-1	reciprocal of size	53.20	51.96	50.48	50.56	50.00	50.00	50.00	50.72	51.52	53.20
-0.5		54.00	51.20	50.08	50.00	50.00	50.12	50.00	51.04	52.36	52.48
0	random	53.44	51.12	51.04	50.72	50.24	50.48	51.04	50.88	51.72	51.88
0.5		52.72	52.12	51.44	50.96	50.80	50.24	51.38	51.84	51.64	51.08
1	propotion to size	54.12	52.44	51.04	51.76	50.64	51.12	50.84	50.40	51.04	50.00
2		53.74	53.64	52.24	51.76	51.44	50.48	50.36	50.32	50.92	50.60
20	biggest first	54.68	55.36	54.88	55.16	54.48	53.76	53.68	53.12	53.00	54.00

n3		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	55.00	54.56	53.96	54.16	54.24	53.52	53.24	53.56	54.62	54.84
-2		53.18	52.66	52.68	51.70	51.88	51.88	52.12	52.56	53.40	55.16
-1	reciprocal of size	53.40	52.40	51.88	51.64	51.54	51.82	51.88	52.50	53.52	54.44
-0.5		53.04	51.64	51.64	51.68	51.72	52.00	52.44	52.60	53.24	53.68
0	random	53.32	52.20	51.64	51.74	51.98	52.82	52.90	53.46	53.02	52.76
0.5		53.08	51.96	52.00	52.04	52.88	53.20	53.44	53.28	52.88	52.18
1	propotion to size	54.02	52.96	52.80	53.36	53.32	53.44	53.16	52.76	52.12	52.00
2		54.14	53.12	52.80	53.16	52.88	53.36	52.48	52.08	51.88	51.88
20	biggest first	54.92	55.40	54.88	54.52	54.60	53.76	53.28	52.52	52.20	53.00

n4		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	91.20	90.44	88.36	87.84	86.16	85.68	86.88	85.96	85.40	90.32
-2		86.56	86.44	83.96	83.40	83.00	83.16	82.76	83.44	84.24	88.32
-1	reciprocal of size	85.20	83.66	82.80	82.86	82.96	82.80	82.88	83.64	84.00	87.36
-0.5		84.48	83.52	82.44	82.64	82.68	83.30	83.36	83.72	84.00	87.40
0	random	84.16	83.14	83.20	83.24	83.52	83.92	83.88	84.00	84.26	84.56
0.5		84.66	83.40	83.52	83.64	84.10	84.12	84.44	84.00	84.00	83.40
1	propotion to size	84.44	83.88	83.88	84.00	84.48	84.00	84.00	83.70	83.84	82.92
2		85.80	83.92	84.12	83.80	83.88	83.76	83.56	83.04	83.00	82.00
20	biggest first	91.40	89.24	88.92	88.00	86.58	86.34	84.62	84.68	83.74	87.92

n5		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	108.48	107.96	107.84	107.12	106.88	107.00	106.60	106.36	108.08	108.52
-2		107.00	105.60	104.92	104.92	105.48	105.24	105.64	105.76	107.76	108.96
-1	reciprocal of size	106.48	105.64	105.16	105.40	104.80	105.28	105.76	106.40	106.96	108.88
-0.5		106.20	104.92	105.00	105.40	104.94	105.92	106.06	106.68	106.64	107.68
0	random	106.56	105.64	105.40	105.52	105.88	106.28	107.00	106.40	106.64	106.12
0.5		105.84	105.40	105.76	106.40	107.24	107.32	107.96	106.80	105.88	105.24
1	propotion to size	106.28	105.26	106.46	107.46	107.94	107.28	106.50	105.88	105.60	104.86
2		107.00	105.90	106.12	107.52	106.60	106.24	105.82	105.16	104.80	104.64
20	biggest first	108.92	108.64	108.24	107.64	107.20	106.64	105.88	106.16	105.48	107.00

Average of Minimum
(first drop first add, different location)

n6		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	106.96	107.80	107.08	106.60	105.20	105.28	103.44	104.36	104.72	107.28
-2		105.80	105.04	104.08	102.88	103.72	103.36	103.08	103.36	103.58	106.36
-1	reciprocal of size	103.40	103.92	103.64	102.76	102.80	102.56	102.84	103.00	103.60	105.82
-0.5		104.24	102.88	102.84	102.64	102.80	102.80	103.00	103.00	104.00	105.08
0	random	104.08	102.84	102.64	102.88	102.88	103.12	103.18	103.82	104.02	103.82
0.5		103.60	102.60	102.62	103.36	104.08	104.70	104.28	104.36	103.76	103.00
1	propotion to size	104.10	103.12	103.24	104.56	104.72	104.36	104.00	103.52	103.08	102.52
2		104.88	104.00	103.72	104.20	104.00	103.80	103.28	102.96	102.92	102.00
20	biggest first	107.64	107.20	106.92	106.12	105.60	104.92	103.96	103.36	103.00	103.08

n7		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	118.52	116.72	117.52	117.28	117.88	114.76	111.52	113.48	113.56	116.08
-2		116.28	116.48	116.56	115.24	109.76	105.80	105.04	105.08	107.44	115.34
-1	reciprocal of size	114.46	109.64	108.16	108.04	107.64	103.88	104.16	104.16	107.28	112.70
-0.5		109.10	107.70	106.44	104.94	104.62	103.46	103.86	104.84	108.12	111.56
0	random	107.02	105.76	104.56	105.58	105.48	107.88	109.78	110.12	110.16	109.04
0.5		107.56	105.92	109.00	111.38	111.50	109.80	109.14	108.08	108.08	105.56
1	propotion to size	108.16	108.56	111.98	110.20	108.84	107.28	106.28	106.24	104.92	103.94
2		109.20	108.48	109.44	108.44	105.76	105.08	105.26	105.12	103.76	102.48
20	biggest first	115.80	116.64	115.80	112.44	110.56	110.32	108.80	107.74	110.00	114.62

n8		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	89.60	90.38	89.32	88.04	87.56	85.88	85.84	85.38	87.40	90.42
-2		86.34	84.80	84.48	83.68	83.44	83.82	84.44	84.92	86.34	90.12
-1	reciprocal of size	86.06	84.76	84.12	83.60	83.68	83.68	84.36	84.92	85.92	89.96
-0.5		85.20	84.40	83.54	83.96	84.26	84.60	84.90	85.52	86.18	87.90
0	random	84.60	84.56	85.00	85.56	86.30	87.28	87.24	87.30	86.68	85.72
0.5		84.92	85.80	87.04	88.68	89.18	88.68	87.44	86.36	85.30	84.10
1	propotion to size	85.44	86.52	88.86	88.98	88.44	87.50	86.54	85.66	84.70	83.66
2		86.16	86.52	88.24	87.56	86.92	86.12	85.44	84.78	83.80	83.64
20	biggest first	91.12	90.00	88.62	87.40	86.32	85.76	85.12	84.56	83.88	84.64

n9		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	162.62	163.00	161.44	160.98	160.08	159.50	157.82	157.20	157.36	161.68
-2		158.56	156.82	154.96	155.96	155.50	154.66	154.38	154.82	156.00	161.70
-1	reciprocal of size	157.40	155.16	153.88	154.54	153.92	154.24	154.50	154.90	156.54	162.06
-0.5		155.72	154.96	153.96	154.36	154.48	154.76	155.10	156.24	157.04	161.04
0	random	155.52	154.56	154.54	155.48	156.06	157.12	159.20	159.64	158.40	156.46
0.5		155.46	155.06	157.80	160.22	160.80	159.88	159.88	157.98	156.20	154.46
1	propotion to size	156.72	157.70	160.16	161.10	160.48	158.80	158.20	156.32	154.90	153.68
2		157.76	157.50	157.68	157.56	156.88	156.10	155.24	154.66	153.98	153.26
20	biggest first	161.60	161.12	160.10	158.40	156.76	155.64	154.42	153.68	153.18	154.36

n10		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	158.56	158.78	158.66	158.36	157.78	158.22	156.36	156.26	156.14	158.10
-2		157.52	158.00	157.40	155.88	156.76	154.78	153.54	153.50	154.00	159.08
-1	reciprocal of size	157.00	156.20	156.00	155.00	153.76	154.10	153.56	153.88	154.04	158.72
-0.5		156.74	155.20	154.54	154.56	154.26	153.92	154.02	154.24	154.74	159.00
0	random	155.08	155.06	155.32	156.26	156.62	157.30	157.50	157.74	157.30	154.04
0.5		155.82	157.40	157.64	157.92	158.22	157.82	156.20	155.16	154.08	152.60
1	propotion to size	155.94	157.78	157.58	157.32	156.38	155.28	154.28	153.70	152.96	152.08
2		156.16	156.30	155.94	154.82	154.16	153.28	152.68	152.48	152.12	152.00
20	biggest first	158.66	157.72	156.06	155.08	154.08	153.54	153.14	152.88	152.34	152.12

Average of Minimum
(random add, different location)

σ1: percentage of shapes being dropped;
σ2: parameter of size pereference, used in formula (Vi / Vm) ^ σ2 where Vi is the volume of a shape, Vm is the median volume;
Each value is an average over 50 runs;
Highlighted values are the 10 best results for each instance, i.e. the best settings of σ1 and σ2 for each instance.

n2		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	54.34	54.68	53.56	52.32	52.88	51.96	50.16	50.48	50.00	51.40
-2		52.84	52.04	51.44	50.48	50.48	50.00	50.00	50.00	50.40	51.68
-1	reciprocal of size	53.20	51.00	50.84	50.24	50.24	50.00	50.00	50.00	50.80	51.64
-0.5		53.62	51.52	50.48	50.32	50.48	50.00	50.12	50.68	50.00	51.64
0	random	53.28	51.50	51.64	50.68	50.48	50.40	51.08	51.12	51.88	51.32
0.5		53.20	51.80	51.76	50.80	51.24	50.84	51.74	52.22	52.48	51.94
1	propotion to size	53.92	52.72	52.24	51.52	52.00	52.48	52.88	52.72	52.78	52.12
2		53.94	53.52	52.40	52.40	52.72	52.88	52.76	52.60	52.60	51.60
20	biggest first	55.32	54.72	54.12	54.28	53.60	53.92	53.36	52.64	52.04	52.08

n3		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	55.56	54.52	53.84	54.08	52.64	52.28	52.00	52.12	52.36	52.88
-2		53.28	52.92	52.64	52.00	51.76	51.48	51.36	52.00	52.52	52.88
-1	reciprocal of size	53.52	52.40	51.72	51.74	51.44	51.84	51.84	51.88	52.30	52.88
-0.5		52.92	51.88	51.76	51.76	51.72	51.76	52.00	52.64	52.88	53.08
0	random	53.32	52.66	51.70	51.76	52.12	52.40	53.10	53.18	53.24	52.88
0.5		52.76	52.46	52.00	52.60	53.12	53.64	54.12	53.74	53.46	52.76
1	propotion to size	53.16	52.36	52.84	53.06	54.36	53.96	54.00	53.42	53.64	52.88
2		54.28	53.66	53.30	53.84	54.04	54.34	54.10	53.76	53.04	52.88
20	biggest first	55.56	54.80	54.28	54.48	53.68	54.00	54.00	53.52	53.06	52.72

n4		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	90.64	91.04	90.60	86.76	87.52	86.84	85.56	83.12	83.52	84.64
-2		87.42	84.88	84.06	83.34	82.90	82.76	82.74	83.02	83.58	84.20
-1	reciprocal of size	85.44	83.64	83.20	82.92	82.52	82.80	83.00	83.20	83.76	84.60
-0.5		84.80	83.44	83.20	83.04	82.72	83.10	83.36	83.38	83.86	84.22
0	random	84.26	83.48	83.52	83.38	83.60	84.00	84.00	83.88	84.62	84.00
0.5		83.80	83.20	83.32	83.88	84.00	84.72	85.24	85.32	84.56	84.48
1	propotion to size	84.68	83.96	84.24	85.60	85.72	85.92	84.64	85.04	84.52	84.28
2		85.66	84.36	85.58	86.14	86.06	85.70	85.32	84.48	84.08	84.32
20	biggest first	91.08	86.34	86.80	85.94	85.14	85.00	84.46	84.88	84.50	84.58

n5		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	108.20	108.16	107.08	106.56	106.96	105.80	105.96	105.40	105.00	106.00
-2		106.84	105.84	104.98	105.12	104.70	104.34	104.74	104.88	105.68	105.88
-1	reciprocal of size	106.40	105.96	105.00	104.96	104.80	104.60	105.36	105.24	105.88	106.12
-0.5		106.06	104.96	105.20	105.30	105.38	105.28	105.88	105.88	106.44	106.16
0	random	105.60	105.40	105.24	105.48	106.08	106.64	107.02	106.64	106.28	106.12
0.5		105.40	105.24	105.96	107.12	108.12	108.16	108.32	107.52	107.28	106.12
1	propotion to size	107.04	106.12	107.06	108.38	108.16	108.68	108.12	107.28	106.60	106.24
2		107.70	107.16	107.78	108.56	108.16	108.44	108.12	107.36	106.76	106.28
20	biggest first	108.90	107.80	108.76	108.96	108.28	108.00	107.28	107.00	106.36	106.12

Average of Minimum
(random add, different location)

n6		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	107.96	107.48	106.04	106.52	105.08	104.84	103.52	103.20	103.24	103.64
-2		105.96	104.32	103.68	103.72	103.24	102.86	102.72	102.88	103.00	103.96
-1	reciprocal of size	104.36	102.88	103.24	102.60	102.84	102.64	102.50	102.86	103.00	103.82
-0.5		104.36	103.00	102.84	102.68	102.36	102.76	102.76	103.00	103.32	103.76
0	random	103.44	102.66	102.80	102.66	102.86	103.00	103.12	103.78	103.52	103.84
0.5		103.64	102.96	102.88	103.20	104.48	104.48	104.52	104.88	104.60	103.64
1	propotion to size	103.84	103.24	103.36	104.88	105.08	105.28	105.04	104.56	103.88	103.68
2		105.08	104.04	104.44	105.42	105.34	104.88	104.88	104.60	104.24	103.76
20	biggest first	107.16	105.20	105.52	105.68	105.32	105.88	104.88	104.60	103.64	103.72

n7		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	116.68	119.06	117.36	119.26	115.64	115.48	112.84	110.08	106.64	109.32
-2		115.78	116.88	115.56	112.92	107.96	109.96	105.16	103.24	104.16	109.78
-1	reciprocal of size	114.68	111.80	108.68	107.78	106.66	103.44	103.20	103.42	104.88	109.02
-0.5		109.56	109.20	106.60	103.68	103.72	104.00	103.36	104.26	106.24	108.88
0	random	107.96	106.12	104.62	105.32	105.74	107.98	108.86	110.00	109.64	109.16
0.5		105.44	106.30	109.24	112.32	112.56	112.10	110.94	111.66	109.70	109.60
1	propotion to size	108.04	110.62	112.46	112.76	111.04	110.72	110.84	110.76	109.82	108.96
2		110.96	113.38	113.84	111.52	111.76	110.84	110.12	109.16	109.88	108.64
20	biggest first	114.48	113.52	112.04	111.44	110.68	110.28	109.72	109.96	110.24	108.72

n8		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	90.48	90.00	89.64	87.28	86.64	85.60	84.72	84.12	84.36	85.76
-2		87.16	85.08	84.52	84.08	83.48	83.10	83.54	84.08	84.22	85.78
-1	reciprocal of size	85.24	85.56	83.60	83.56	83.36	83.64	84.02	84.28	84.70	85.52
-0.5		85.12	84.00	83.40	83.96	83.98	84.58	84.56	84.84	85.24	86.04
0	random	84.96	84.46	84.80	85.68	86.68	87.20	87.04	87.06	87.00	86.00
0.5		85.02	85.40	87.70	88.66	89.76	89.20	88.46	87.24	87.40	85.72
1	propotion to size	85.16	87.36	89.76	89.52	89.28	88.60	88.30	87.60	87.12	85.88
2		87.04	88.44	89.84	90.14	89.20	88.76	87.92	86.88	86.64	86.00
20	biggest first	90.08	89.42	88.86	89.74	89.02	88.04	87.72	86.80	86.48	85.68

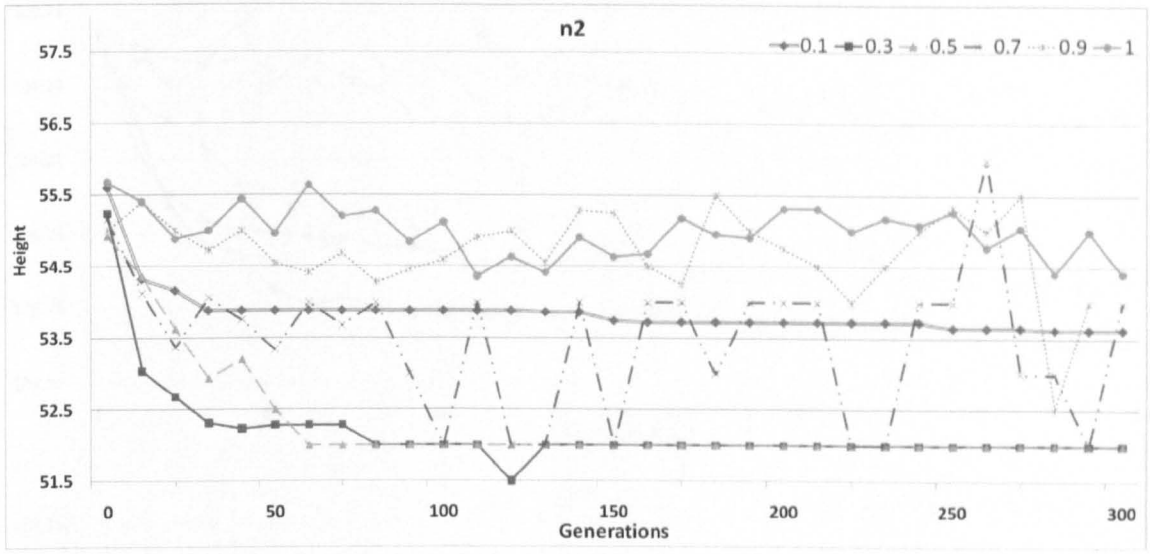
n9		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	162.82	162.08	160.96	161.60	159.32	159.20	157.64	156.80	155.36	156.40
-2		158.48	157.44	155.94	155.90	154.40	154.72	154.10	153.98	155.00	156.92
-1	reciprocal of size	157.36	154.28	155.52	154.52	153.80	153.72	154.12	154.36	155.08	156.70
-0.5		155.48	154.94	154.56	154.08	154.24	154.58	155.28	155.62	156.18	156.72
0	random	155.32	154.10	154.38	154.82	156.14	156.92	158.48	159.18	158.62	157.02
0.5		155.26	155.70	159.02	160.42	161.78	162.22	161.40	159.72	158.96	156.88
1	propotion to size	156.88	158.68	161.40	161.10	161.76	161.04	159.92	159.28	158.62	156.76
2		158.40	161.04	161.84	161.56	161.34	160.54	160.06	159.64	158.28	156.64
20	biggest first	161.82	160.72	161.46	160.94	161.08	160.30	159.20	158.12	157.40	157.04

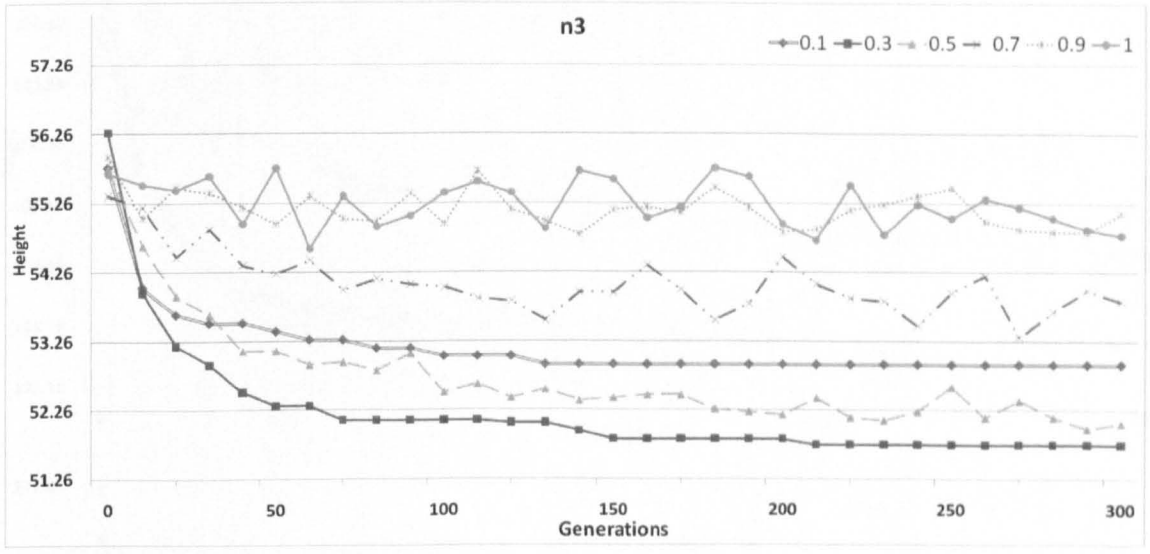
n10		σ1									
σ2		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-20	smallest first	158.64	158.80	159.06	158.36	158.94	158.18	158.18	156.24	154.00	154.40
-2		158.12	156.48	156.86	156.12	155.80	154.72	153.60	152.96	152.92	154.42
-1	reciprocal of size	156.48	156.22	155.48	154.38	154.08	153.68	153.36	153.10	153.20	154.42
-0.5		156.04	155.02	154.90	154.60	154.12	153.96	153.40	153.56	153.76	154.56
0	random	154.68	155.10	155.56	156.28	156.78	156.98	157.28	157.18	156.98	154.42
0.5		155.60	157.36	158.26	158.52	158.36	158.42	157.74	157.00	155.94	154.46
1	propotion to size	156.90	157.86	158.52	158.20	157.88	157.96	157.48	156.86	156.32	154.36
2		157.56	157.58	158.10	158.14	157.44	157.54	156.72	156.64	156.12	154.36
20	biggest first	157.38	158.38	157.56	157.94	157.24	157.02	156.52	155.76	155.42	154.04

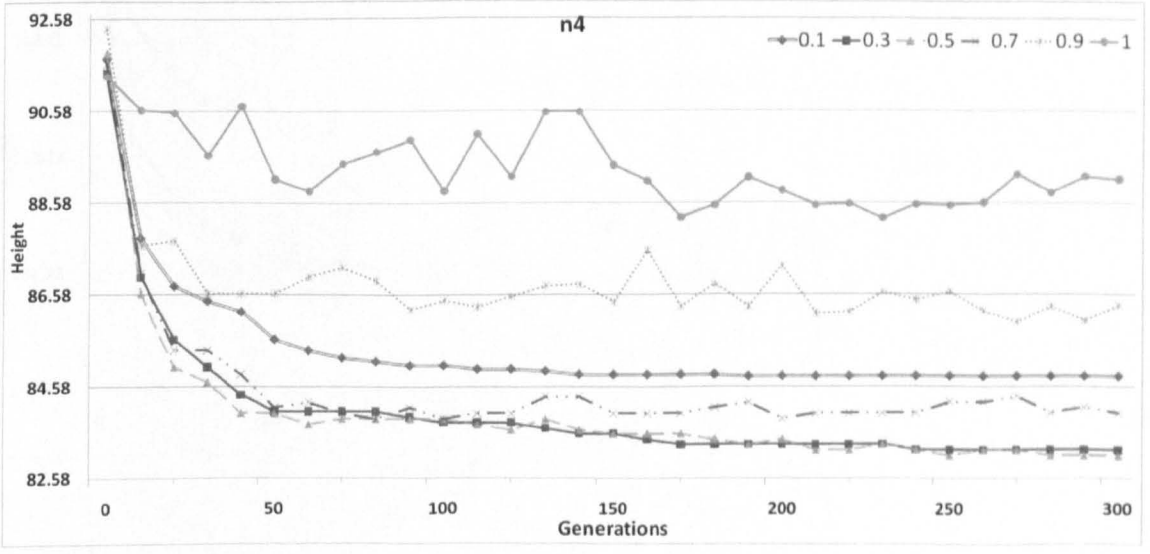
Appendix C

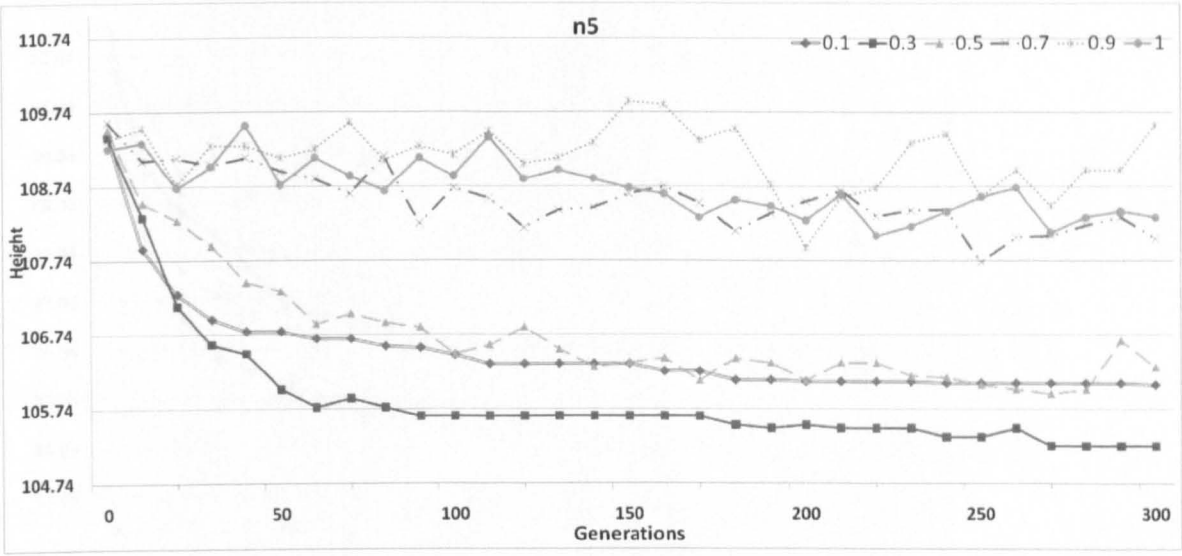
Convergence of ES

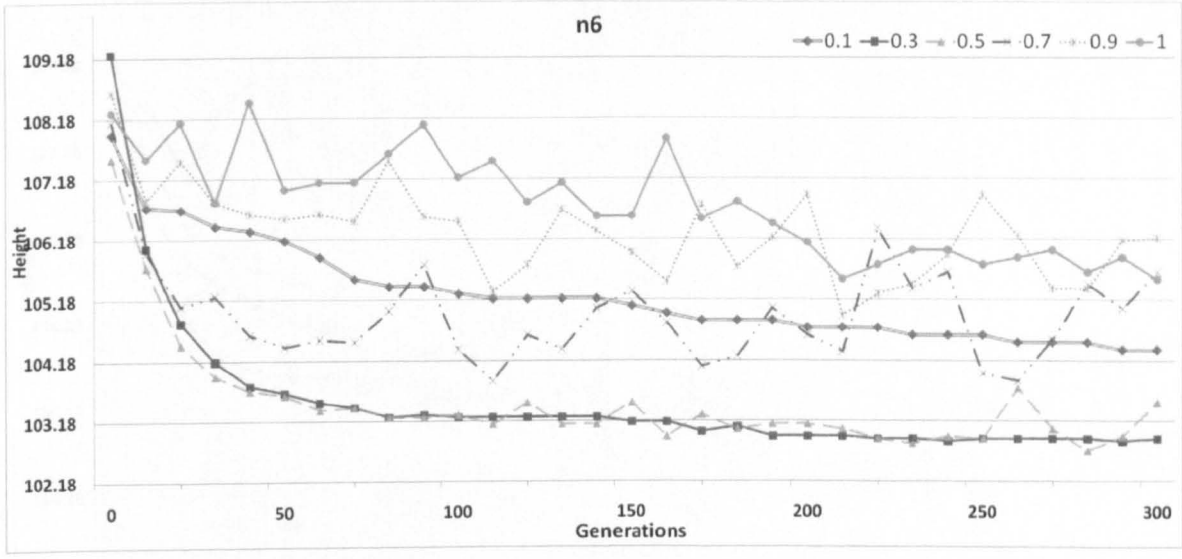
C.1 300 Generations

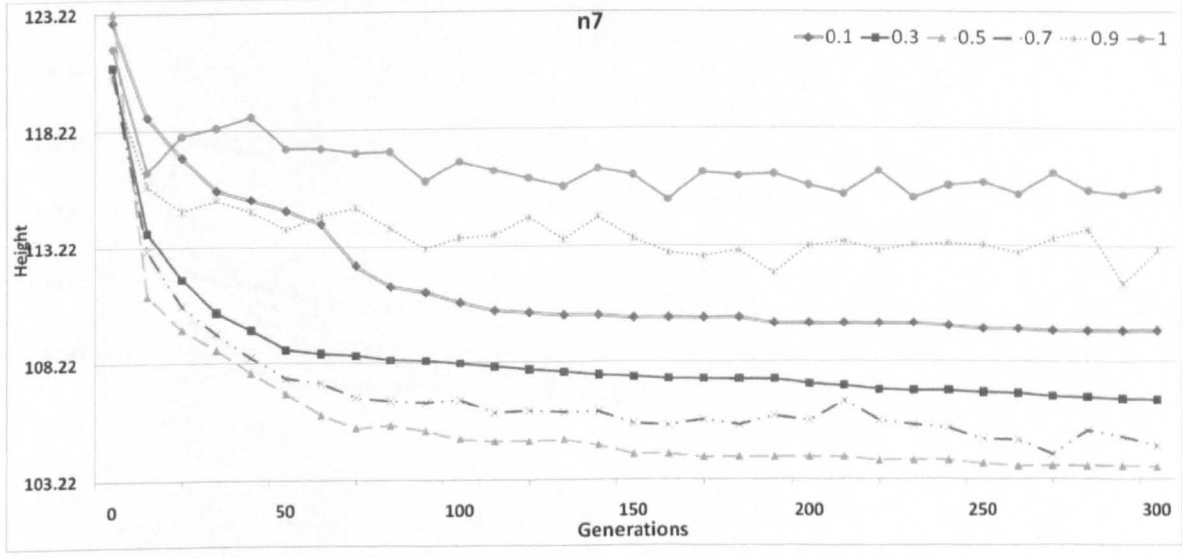


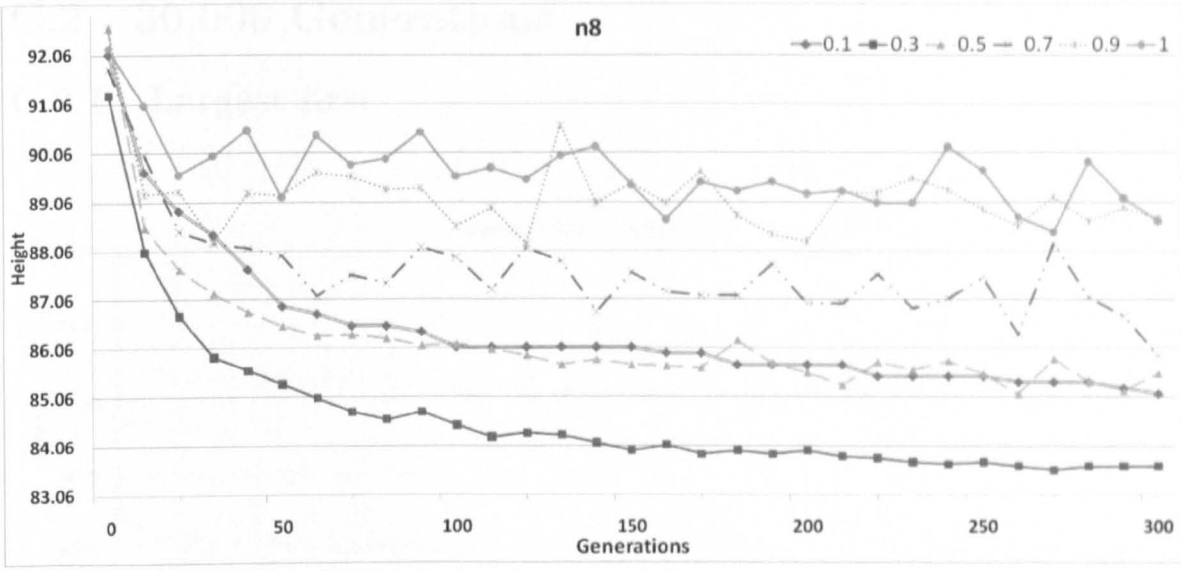


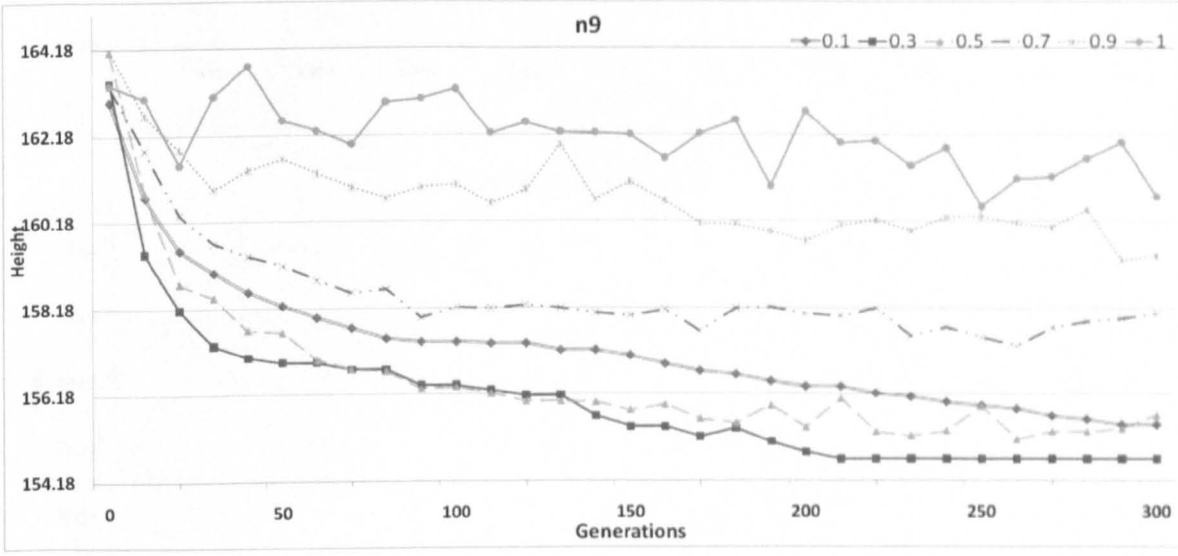


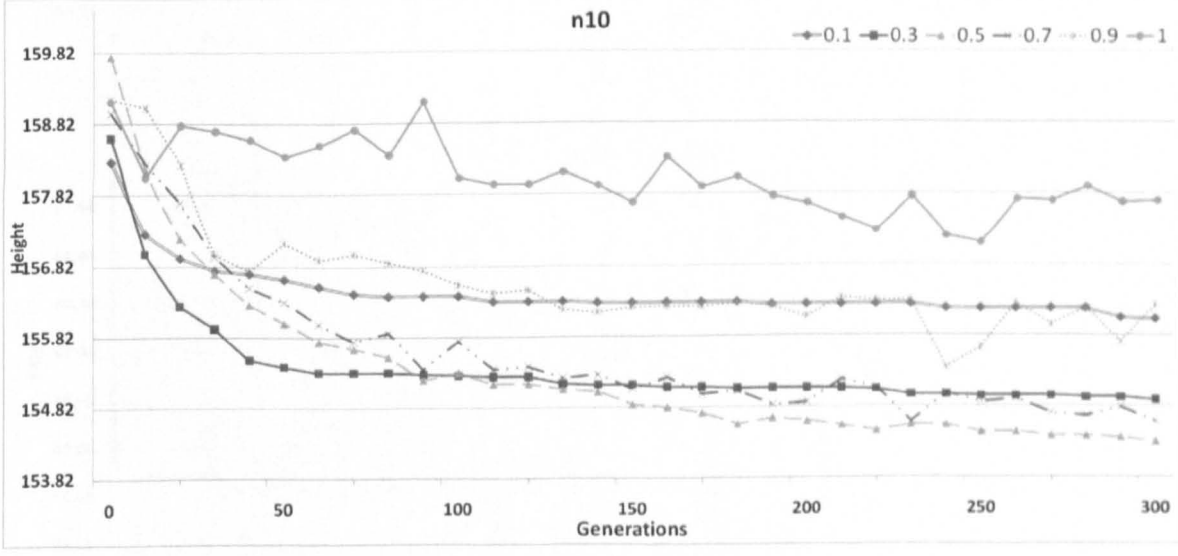

n5
0.1 0.3 0.5 0.7 0.9 1
110.74
109.74
108.74
107.74
106.74
105.74
104.74
Height
0
50
100
150
200
250
300
Generations


n6
0.1 0.3 0.5 0.7 0.9 1
109.18
108.18
107.18
106.18
105.18
104.18
103.18
102.18
Height
0
50
100
150
200
250
300
Generations


n7
0.1 0.3 0.5 0.7 0.9 1
123.22
118.22
113.22
108.22
103.22
Height
0
50
100
150
200
250
300
Generations

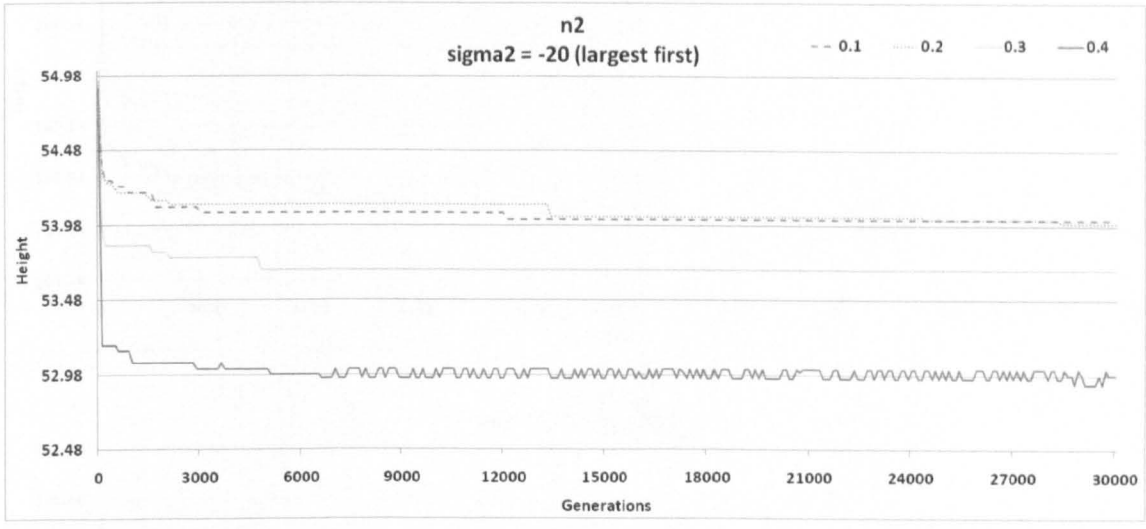

n8
0.1
0.3
0.5
0.7
0.9
1
92.06
91.06
90.06
89.06
88.06
87.06
86.06
85.06
84.06
83.06
Height
0
50
100
150
200
250
300
Generations

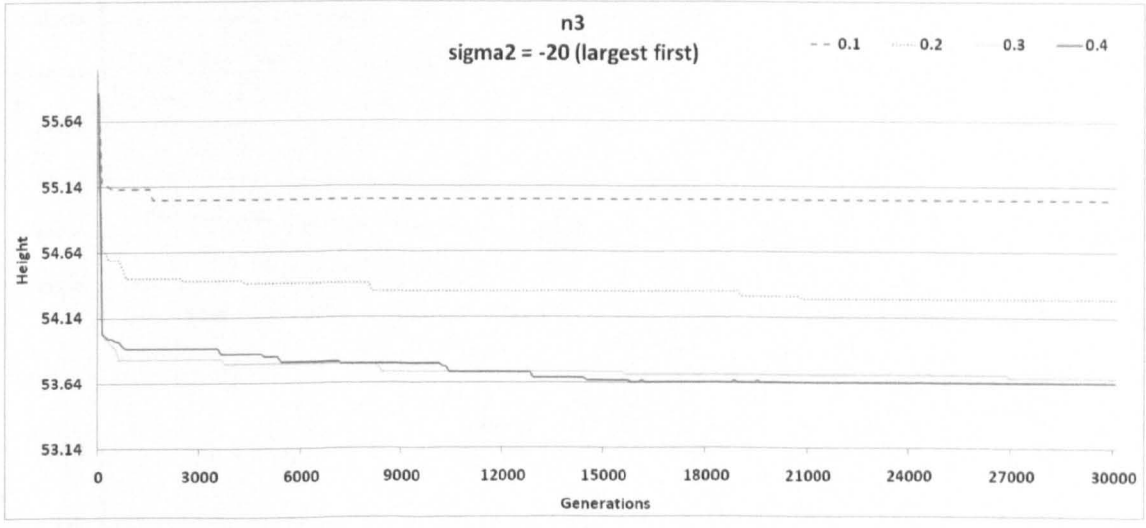

n9
0.1
0.3
0.5
0.7
0.9
1
164.18
162.18
160.18
158.18
156.18
154.18
Height
0
50
100
150
200
250
300
Generations

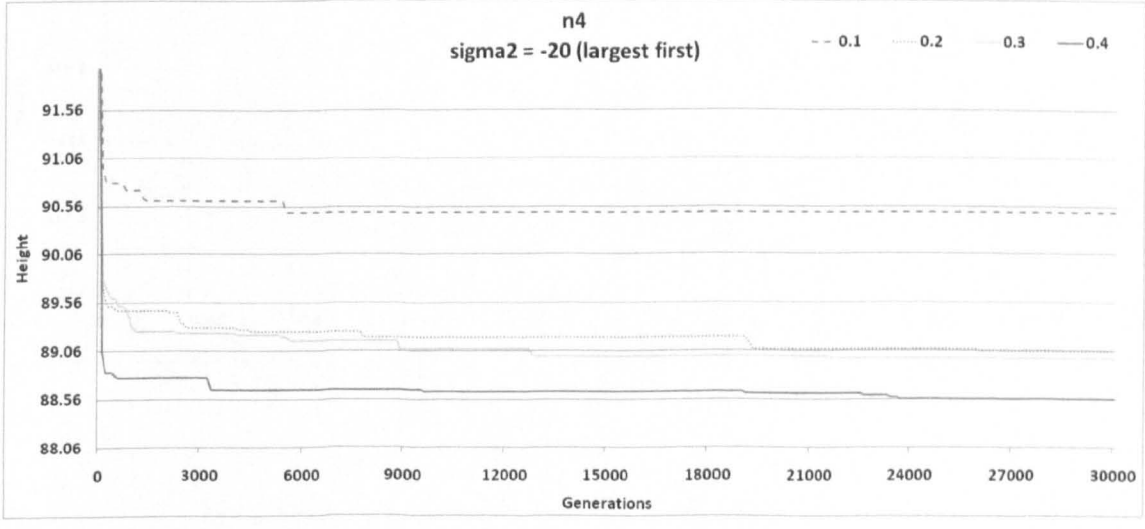

n10
0.1
0.3
0.5
0.7
0.9
1
159.82
158.82
157.82
156.82
155.82
154.82
153.82
Height
0
50
100
150
200
250
300
Generations

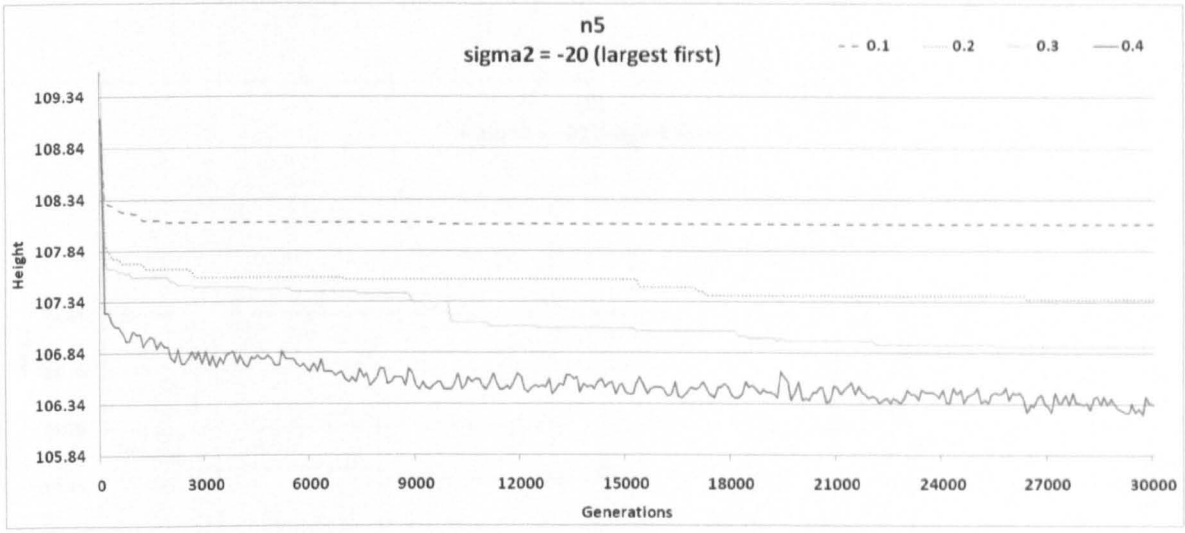
C.2 30,000 Generations

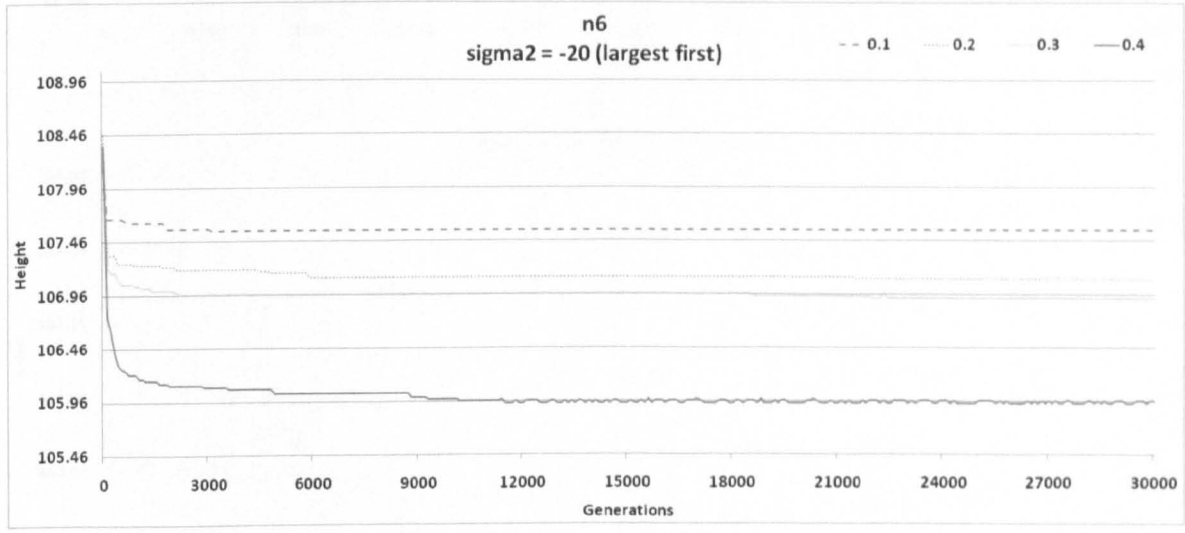
C.2.1 Largest first

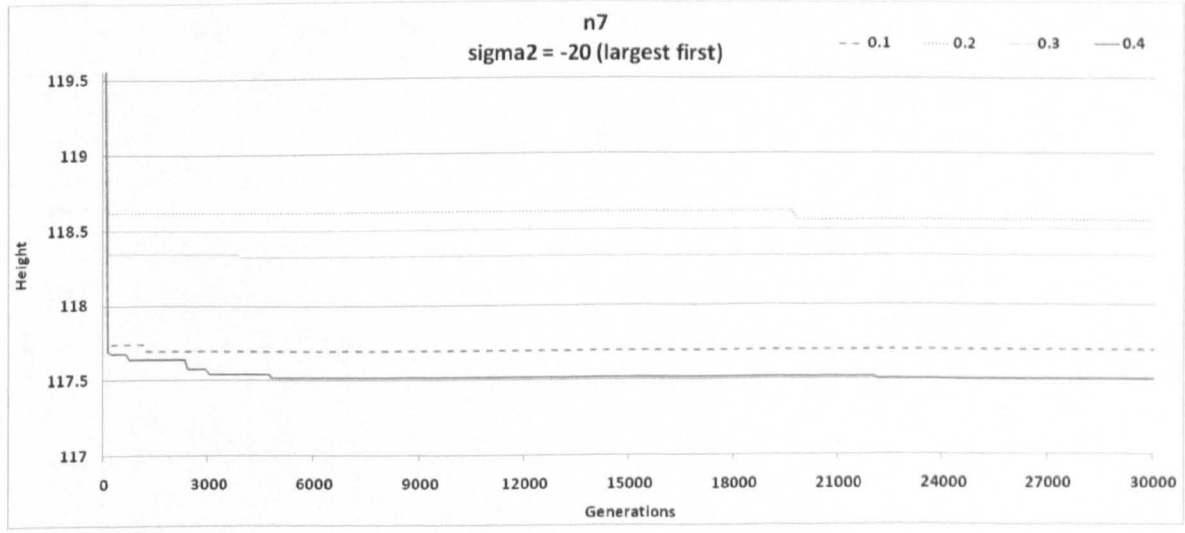


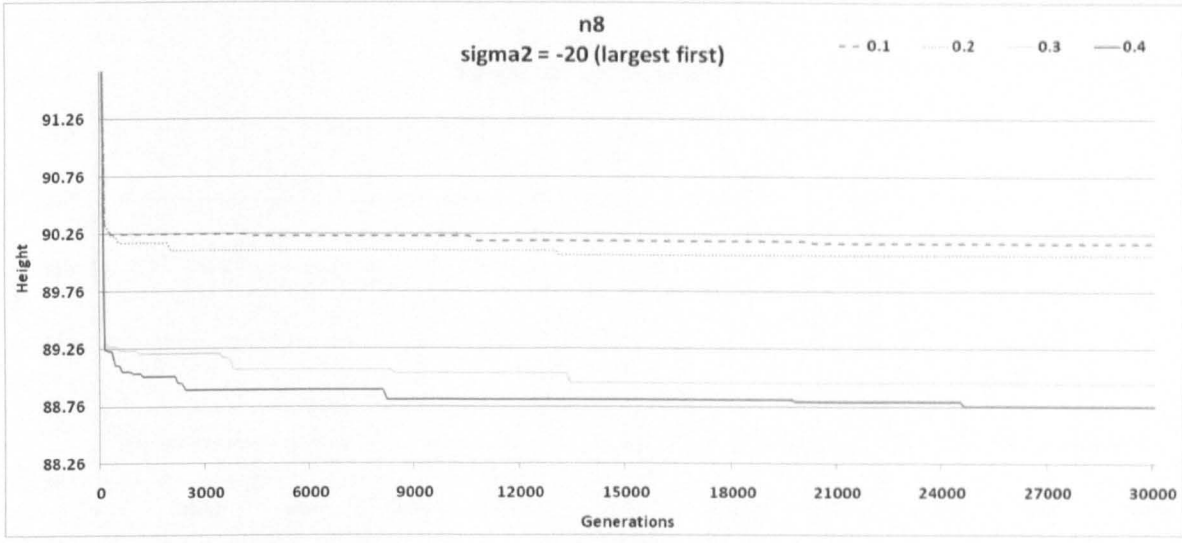


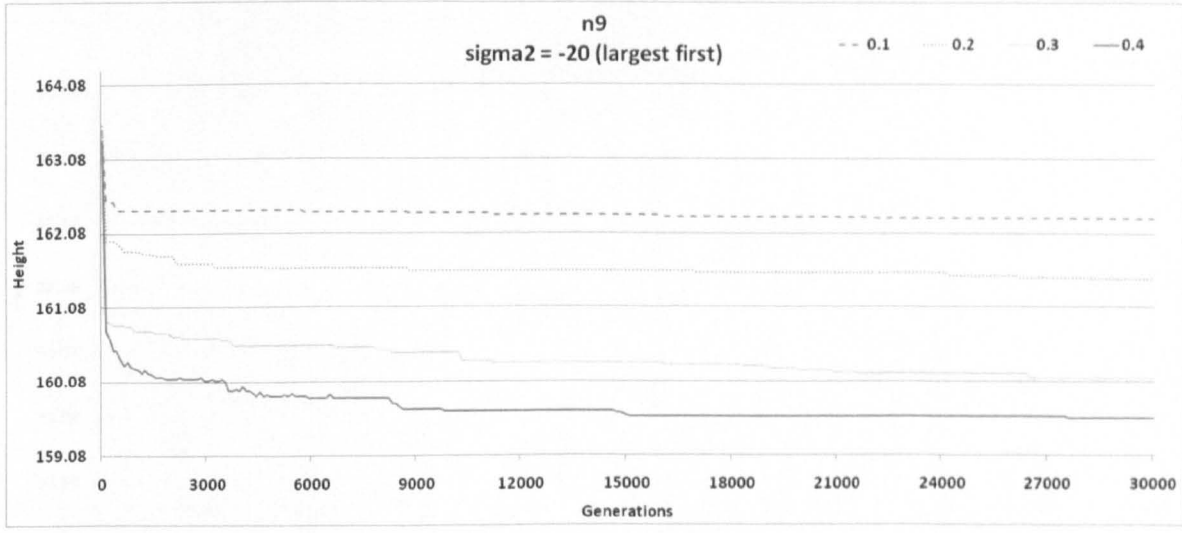


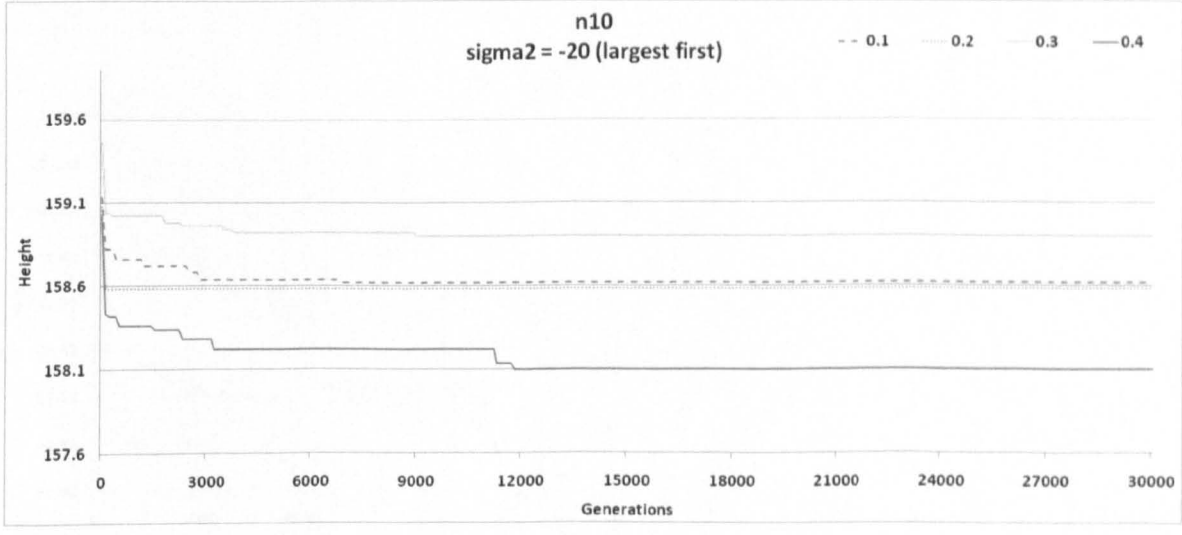

n5
sigma2 = -20 (largest first)
0.1
0.2
0.3
0.4
109.34
108.84
108.34
107.84
107.34
106.84
106.34
105.84
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations


n6
sigma2 = -20 (largest first)
0.1
0.2
0.3
0.4
108.96
108.46
107.96
107.46
106.96
106.46
105.96
105.46
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

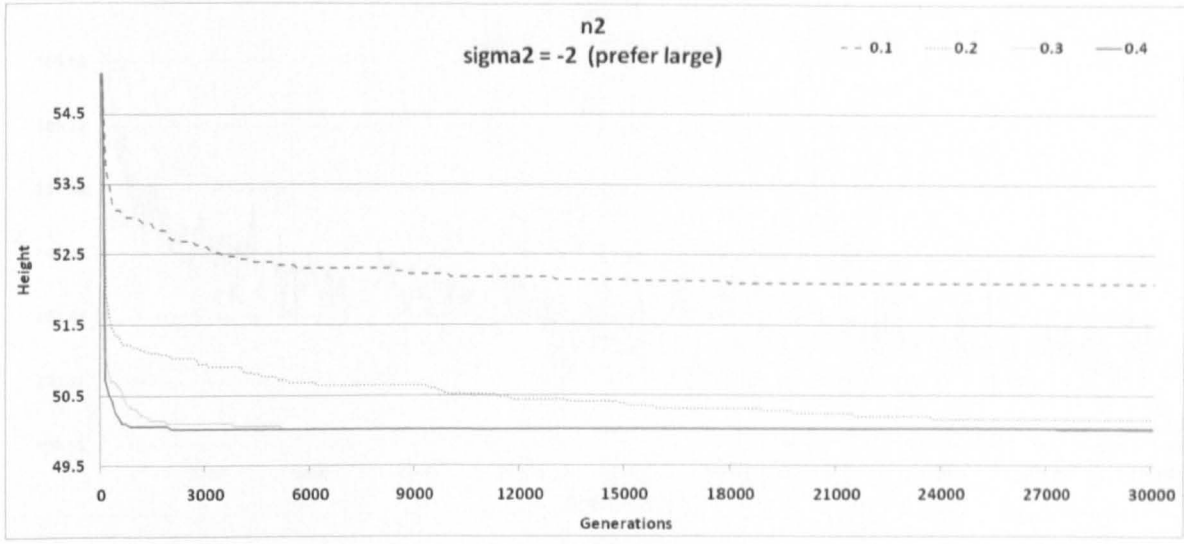

n7
sigma2 = -20 (largest first)
0.1
0.2
0.3
0.4
119.5
119
118.5
118
117.5
117
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

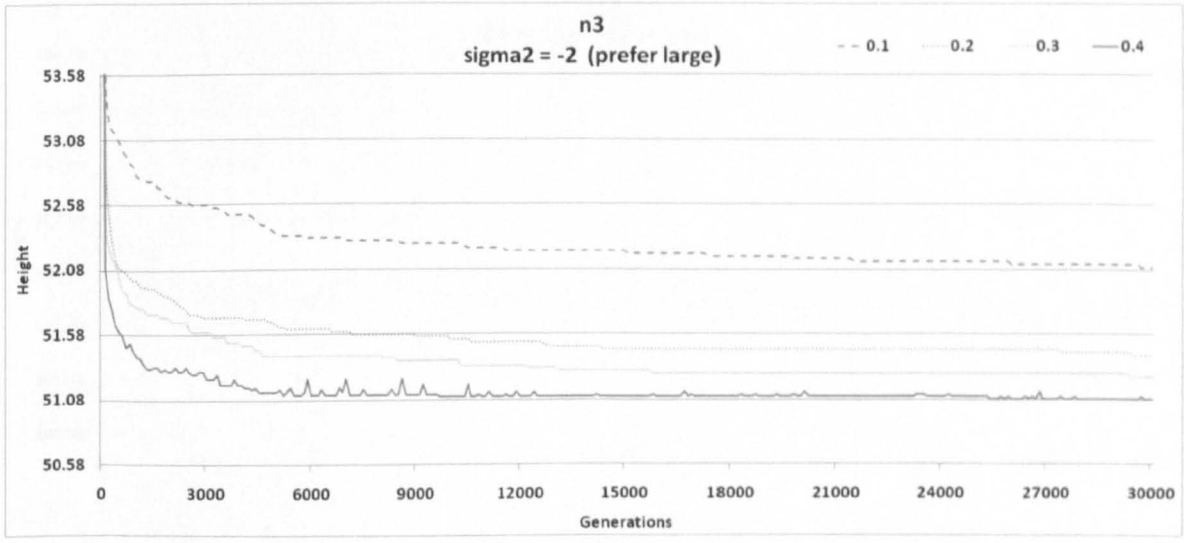

n8
sigma2 = -20 (largest first)
0.1
0.2
0.3
0.4
91.26
90.76
90.26
89.76
89.26
88.76
88.26
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

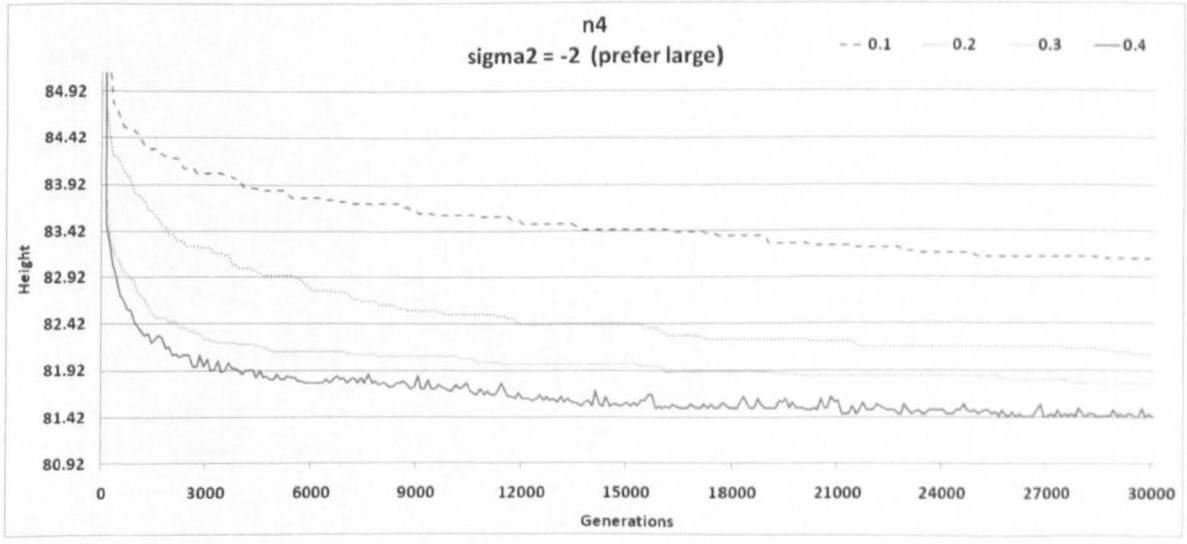

n9
sigma2 = -20 (largest first)
0.1
0.2
0.3
0.4
164.08
163.08
162.08
161.08
160.08
159.08
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

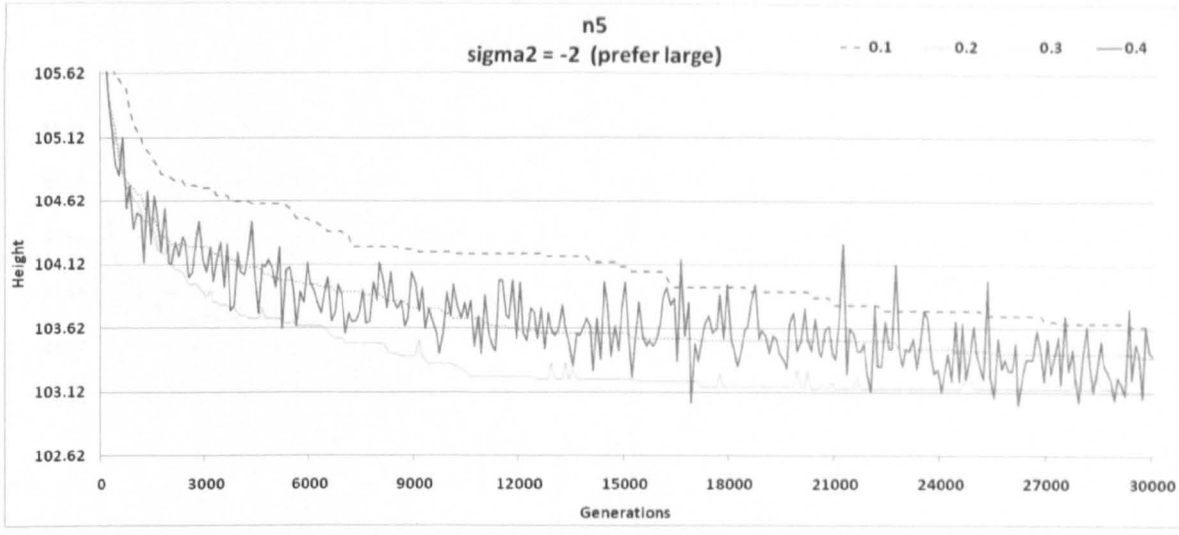

n10
sigma2 = -20 (largest first)
0.1
0.2
0.3
0.4
159.6
159.1
158.6
158.1
157.6
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

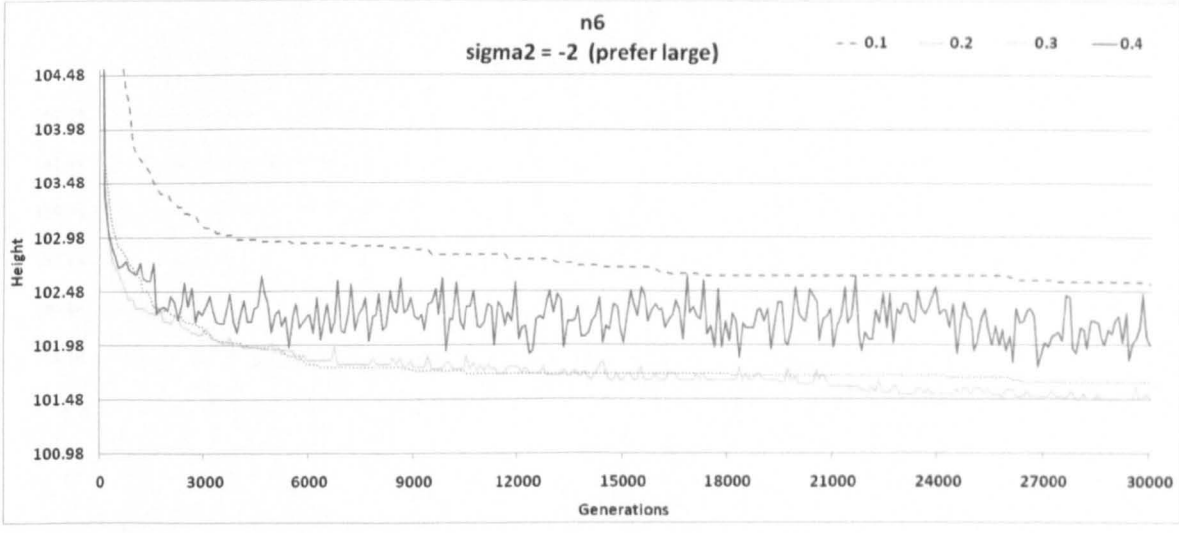
C.2.2 Prefer large

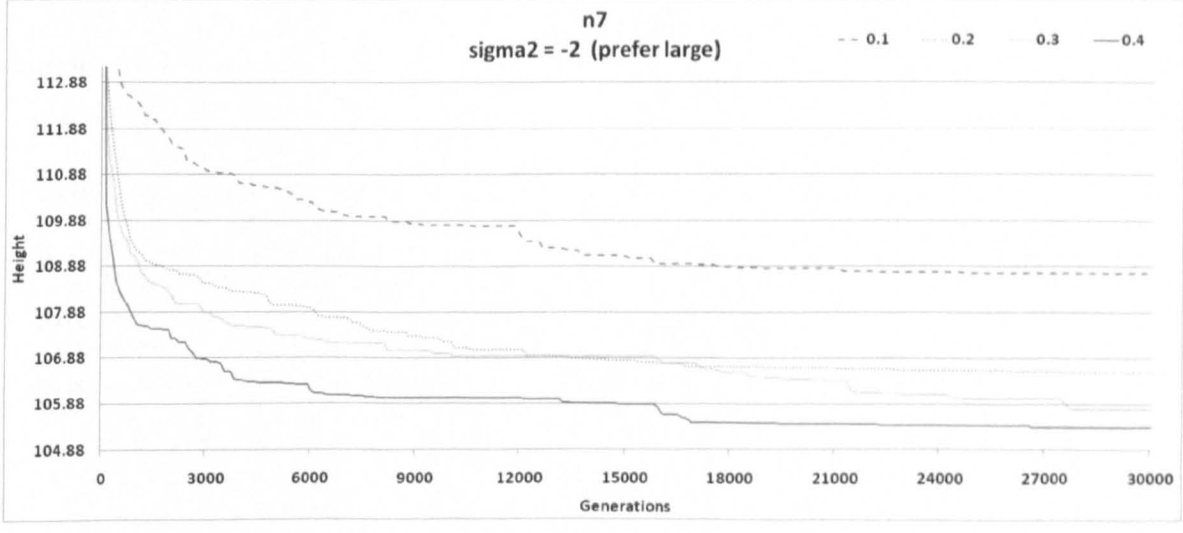


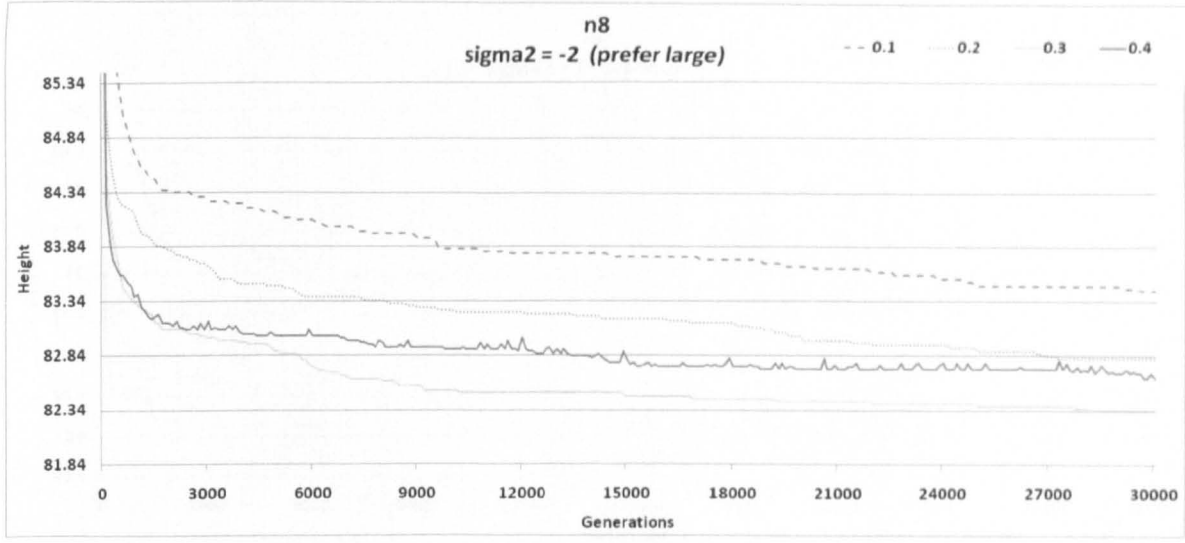


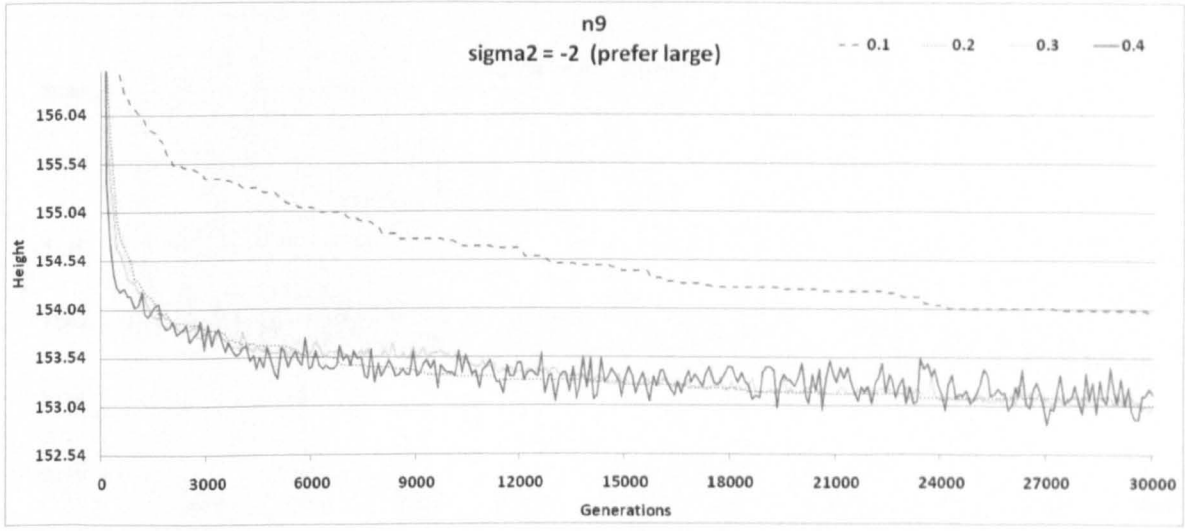


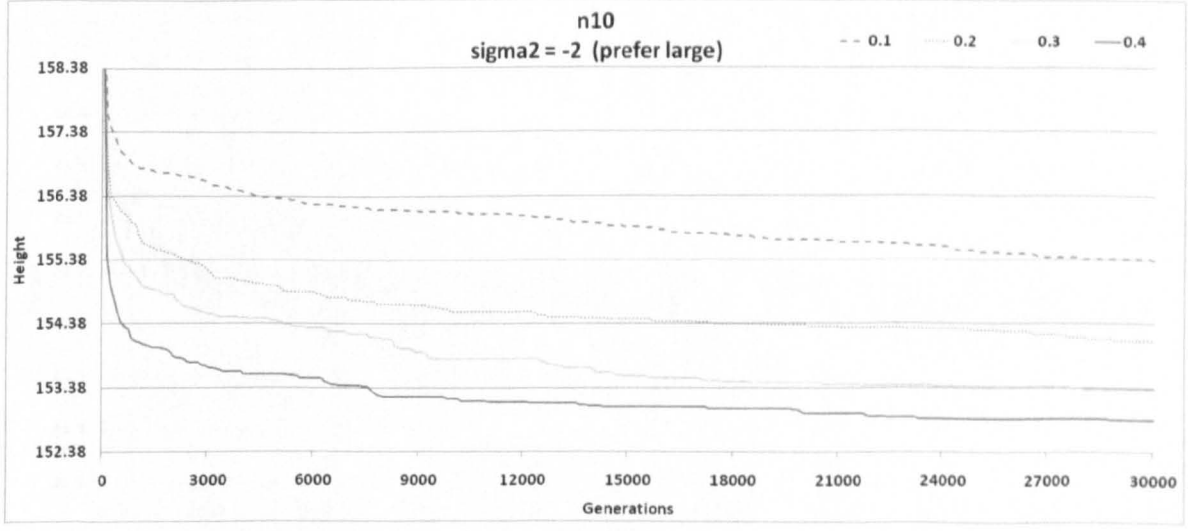

n5
sigma2 = -2 (prefer large)
0.1
0.2
0.3
0.4
105.62
105.12
104.62
104.12
103.62
103.12
102.62
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations


n6
sigma2 = -2 (prefer large)
0.1
0.2
0.3
0.4
104.48
103.98
103.48
102.98
102.48
101.98
101.48
100.98
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

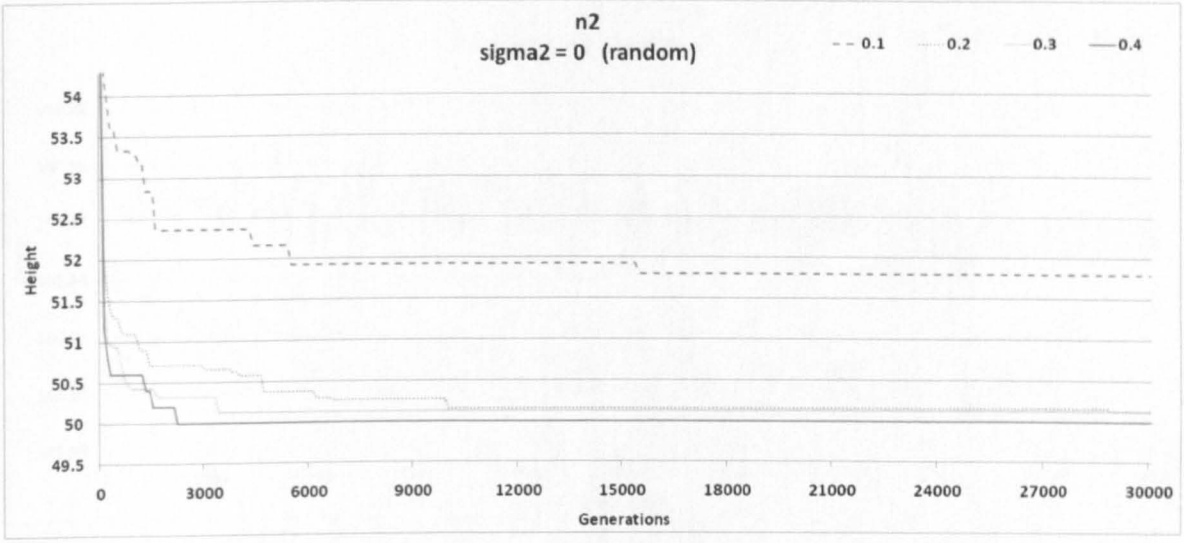

n7
sigma2 = -2 (prefer large)
0.1
0.2
0.3
0.4
112.88
111.88
110.88
109.88
108.88
107.88
106.88
105.88
104.88
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

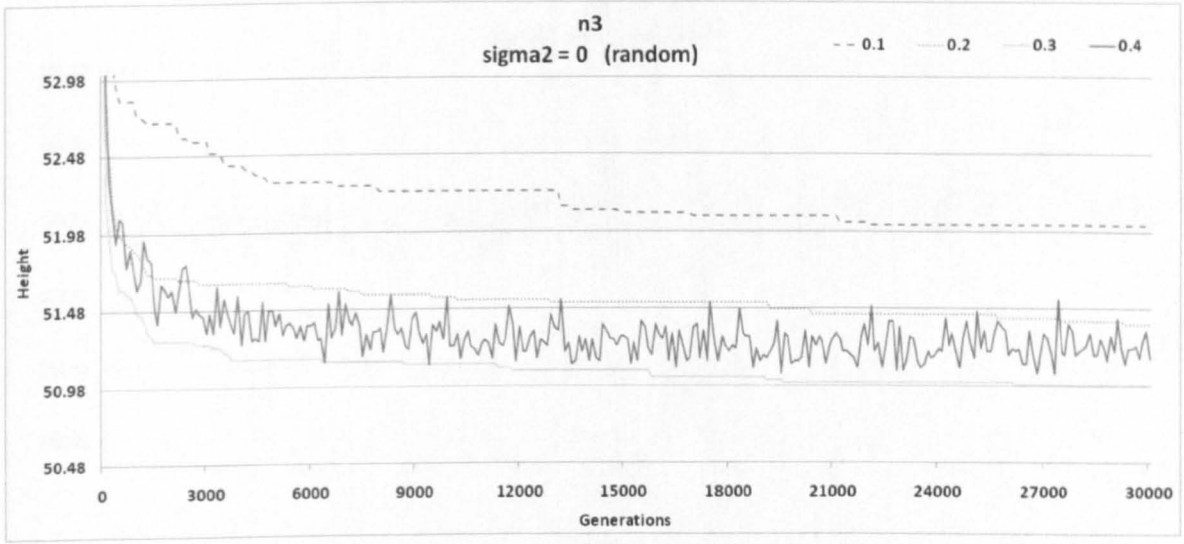

n8
sigma2 = -2 (prefer large)
0.1
0.2
0.3
0.4
85.34
84.84
84.34
83.84
83.34
82.84
82.34
81.84
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

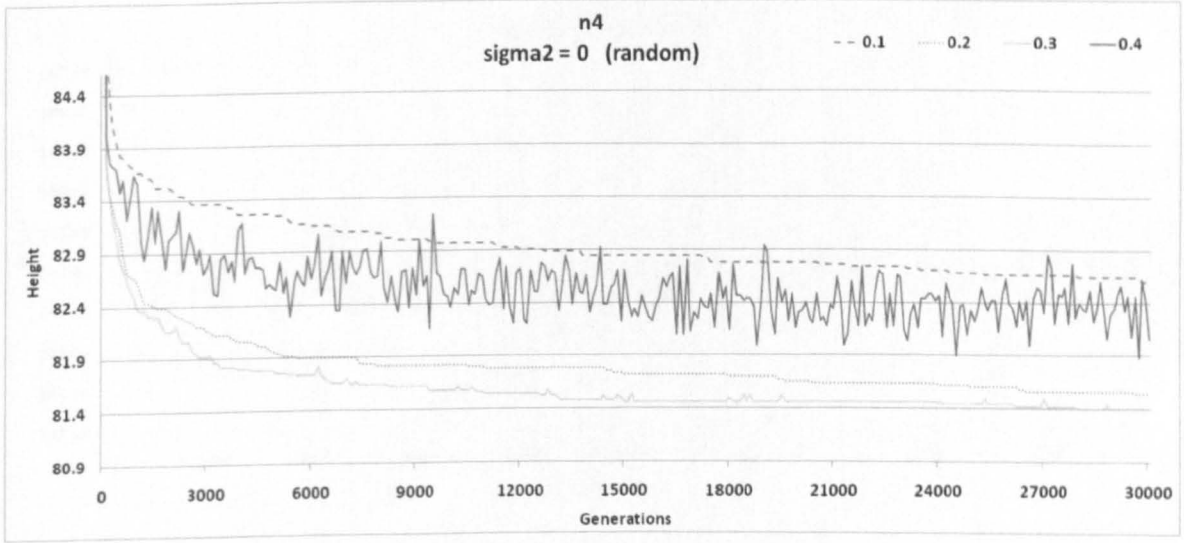

n9
sigma2 = -2 (prefer large)
0.1
0.2
0.3
0.4
156.04
155.54
155.04
154.54
154.04
153.54
153.04
152.54
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

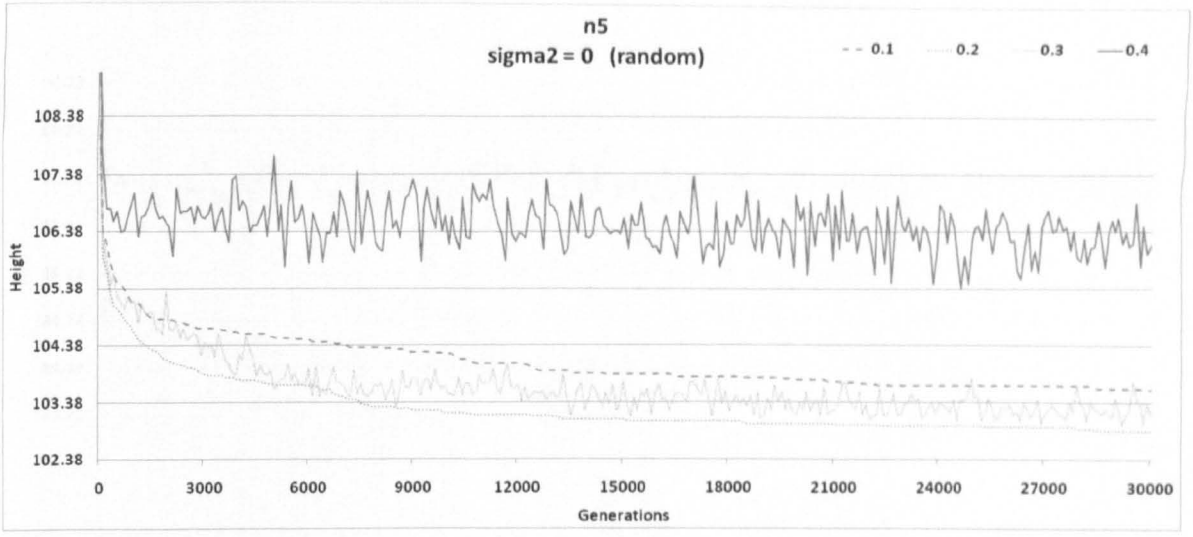

n10
sigma2 = -2 (prefer large)
0.1
0.2
0.3
0.4
158.38
157.38
156.38
155.38
154.38
153.38
152.38
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

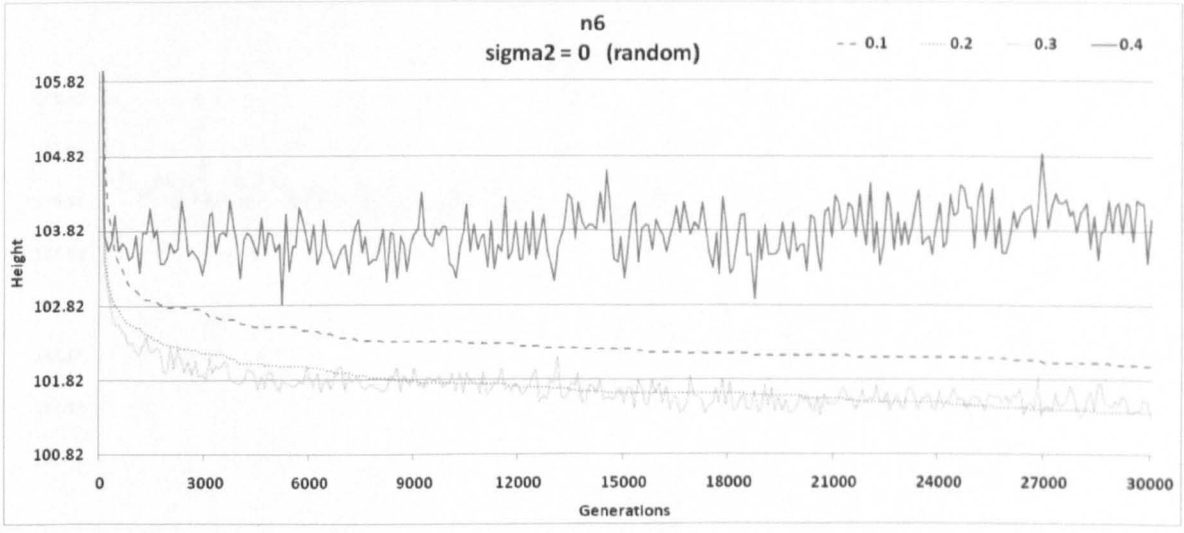
C.2.3 Random

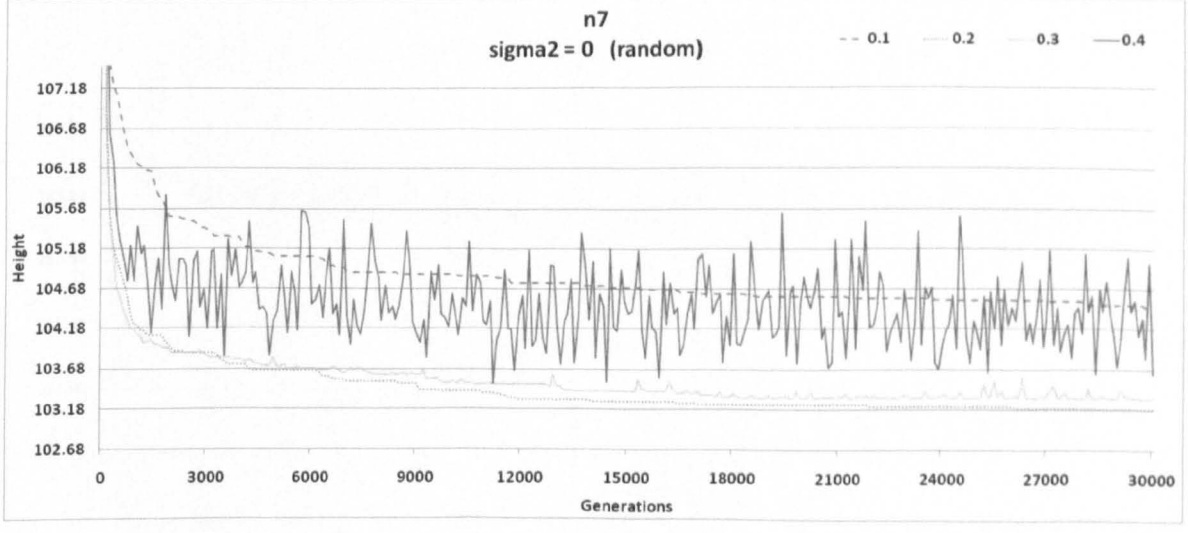

n2
sigma2 = 0 (random)
0.1
0.2
0.3
0.4
Height
54
53.5
53
52.5
52
51.5
51
50.5
50
49.5
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

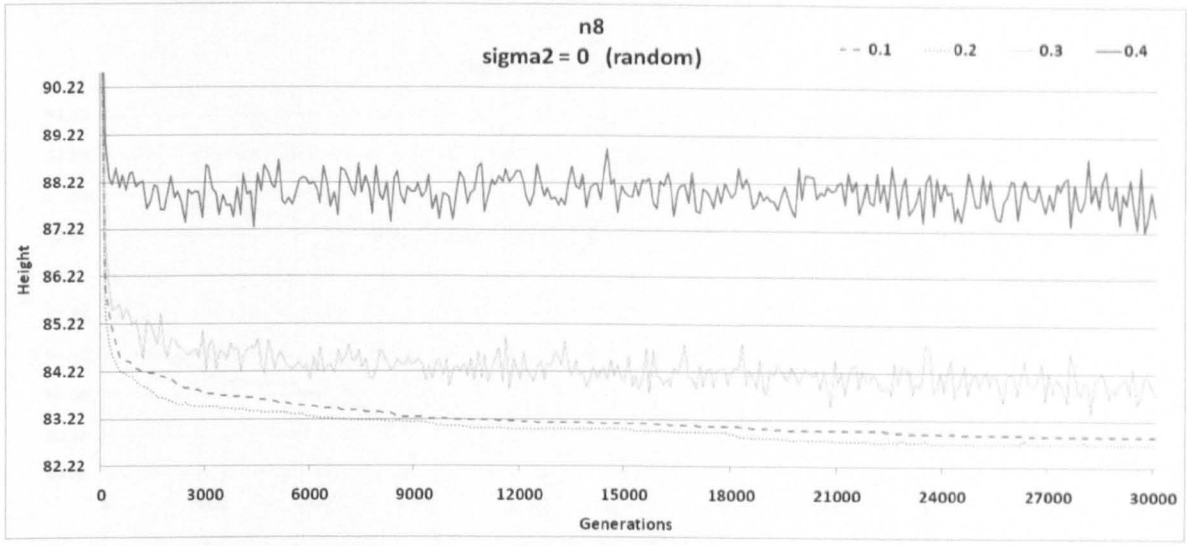

n3
sigma2 = 0 (random)
0.1
0.2
0.3
0.4
Height
52.98
52.48
51.98
51.48
50.98
50.48
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

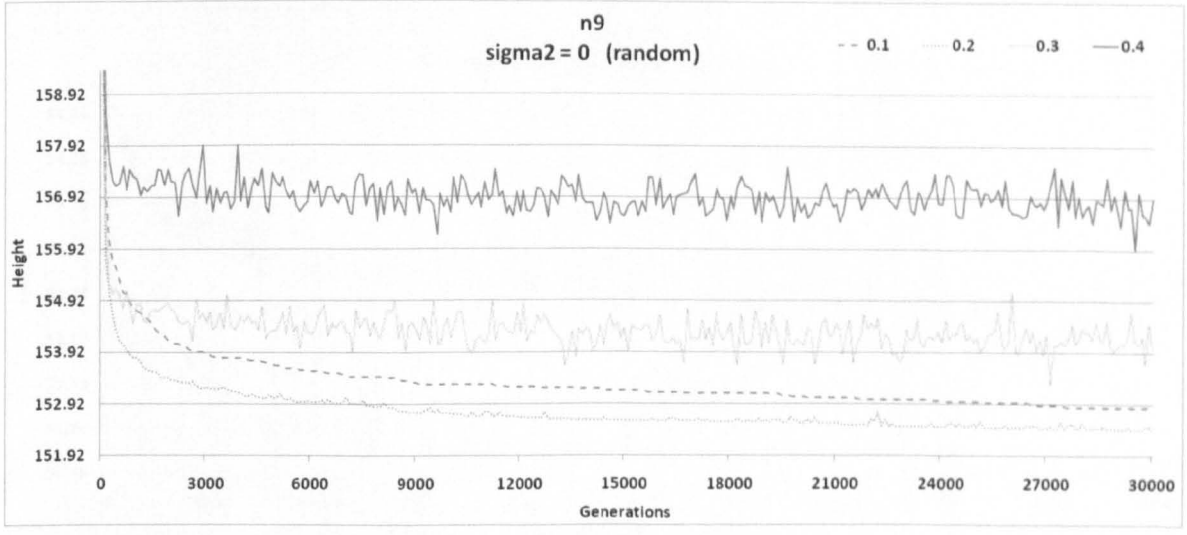

n4
sigma2 = 0 (random)
0.1
0.2
0.3
0.4
Height
84.4
83.9
83.4
82.9
82.4
81.9
81.4
80.9
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

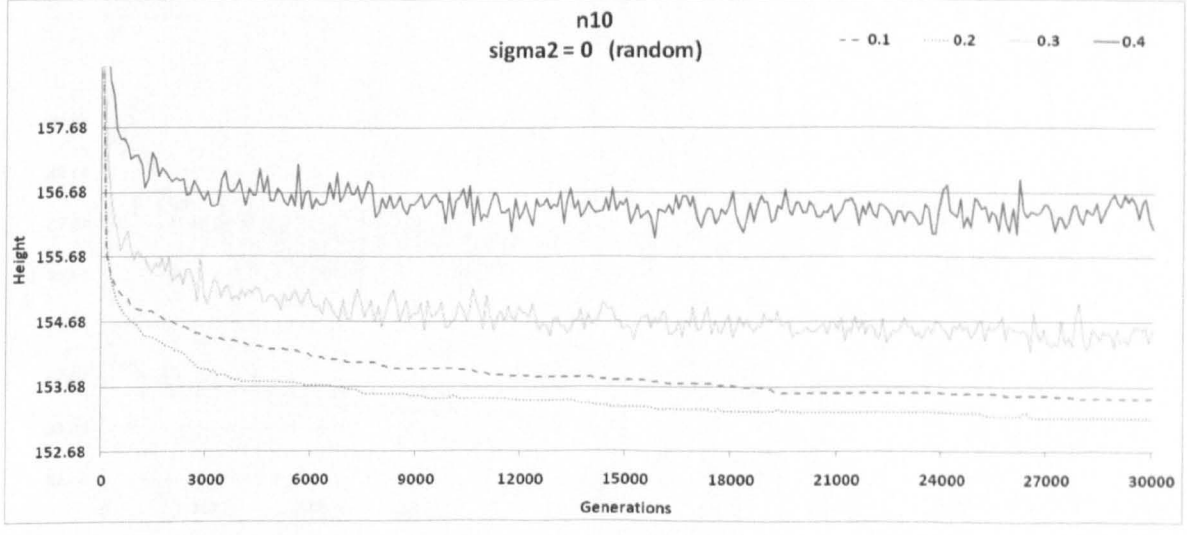

n5
sigma2 = 0 (random)
0.1
0.2
0.3
0.4
108.38
107.38
106.38
105.38
104.38
103.38
102.38
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations


n6
sigma2 = 0 (random)
0.1
0.2
0.3
0.4
105.82
104.82
103.82
102.82
101.82
100.82
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

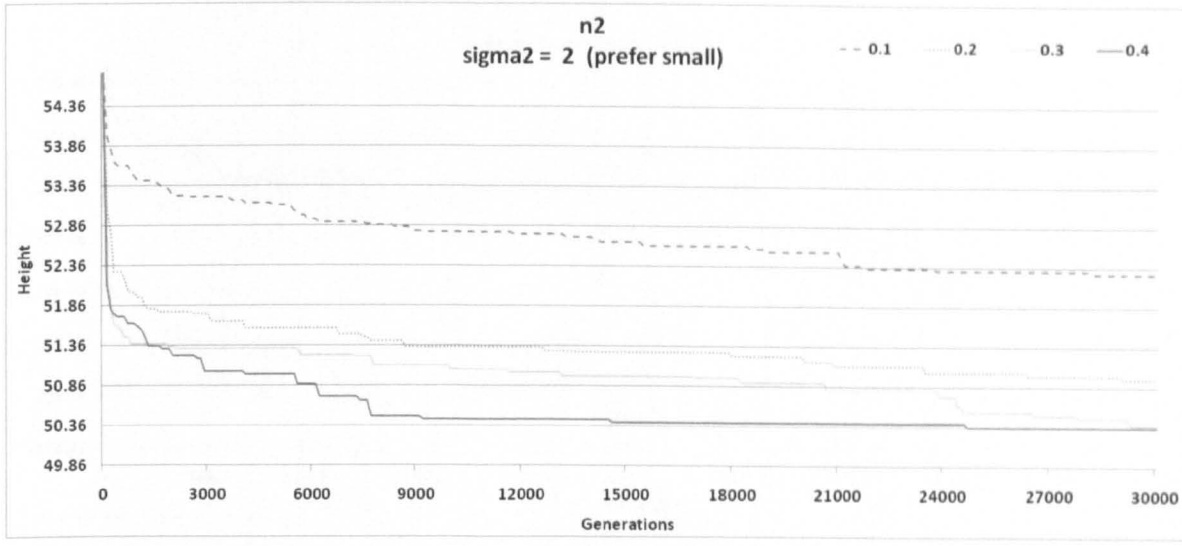

n7
sigma2 = 0 (random)
0.1
0.2
0.3
0.4
107.18
106.68
106.18
105.68
105.18
104.68
104.18
103.68
103.18
102.68
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

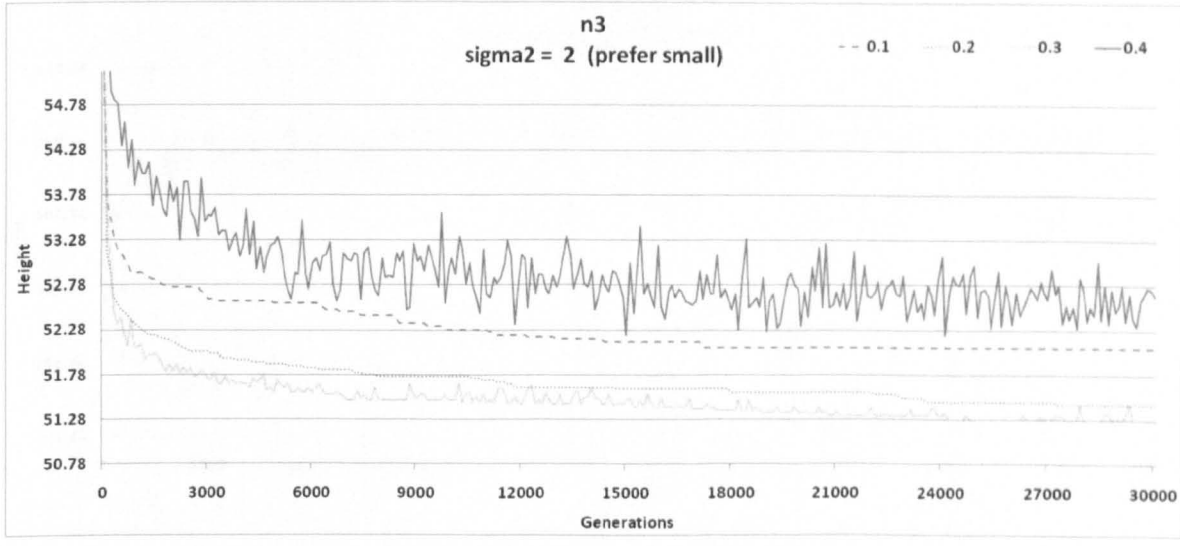

n8
sigma2 = 0 (random)
0.1
0.2
0.3
0.4
90.22
89.22
88.22
87.22
86.22
85.22
84.22
83.22
82.22
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

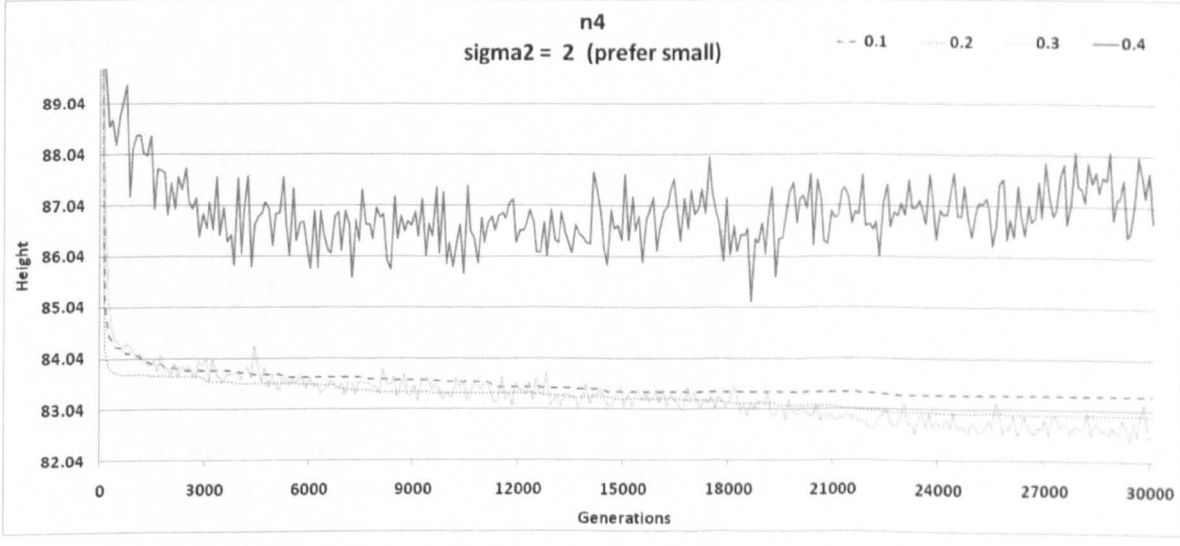

n9
sigma2 = 0 (random)
0.1
0.2
0.3
0.4
158.92
157.92
156.92
155.92
154.92
153.92
152.92
151.92
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

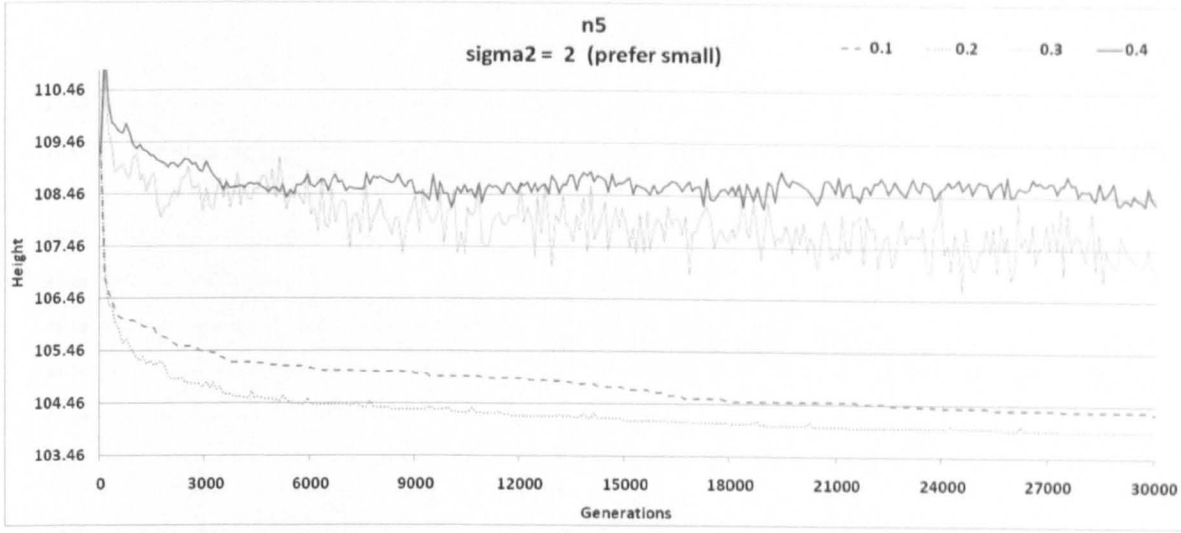

n10
sigma2 = 0 (random)
0.1
0.2
0.3
0.4
157.68
156.68
155.68
154.68
153.68
152.68
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

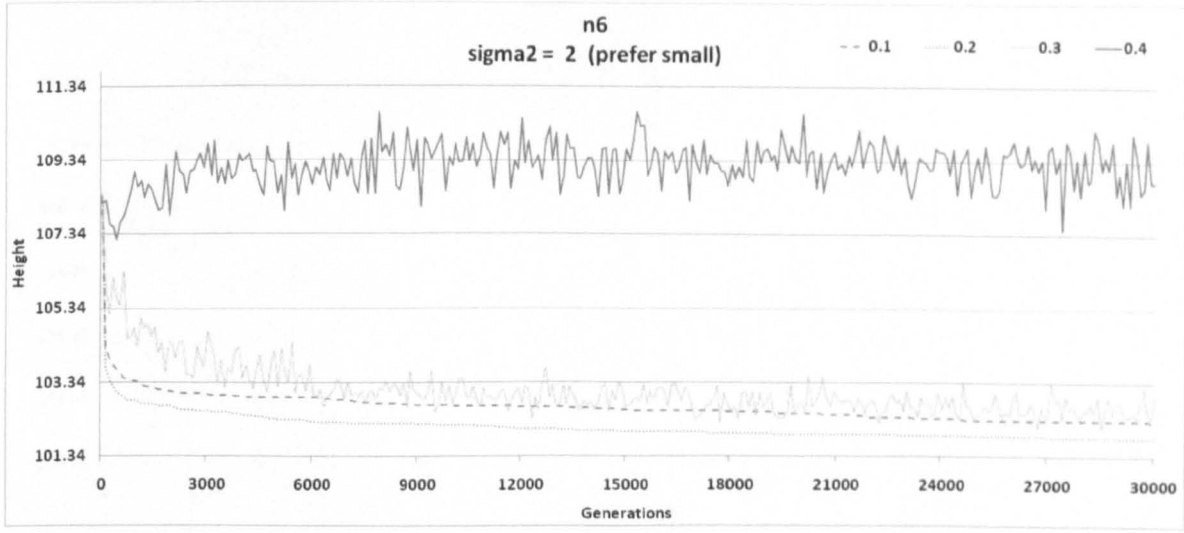
C.2.4 Prefer small

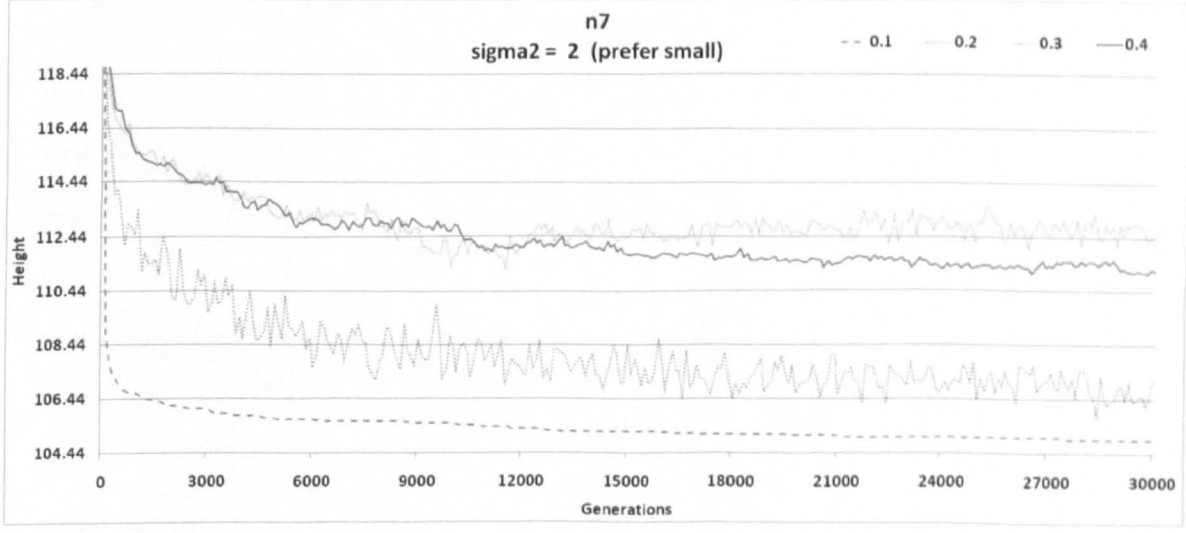


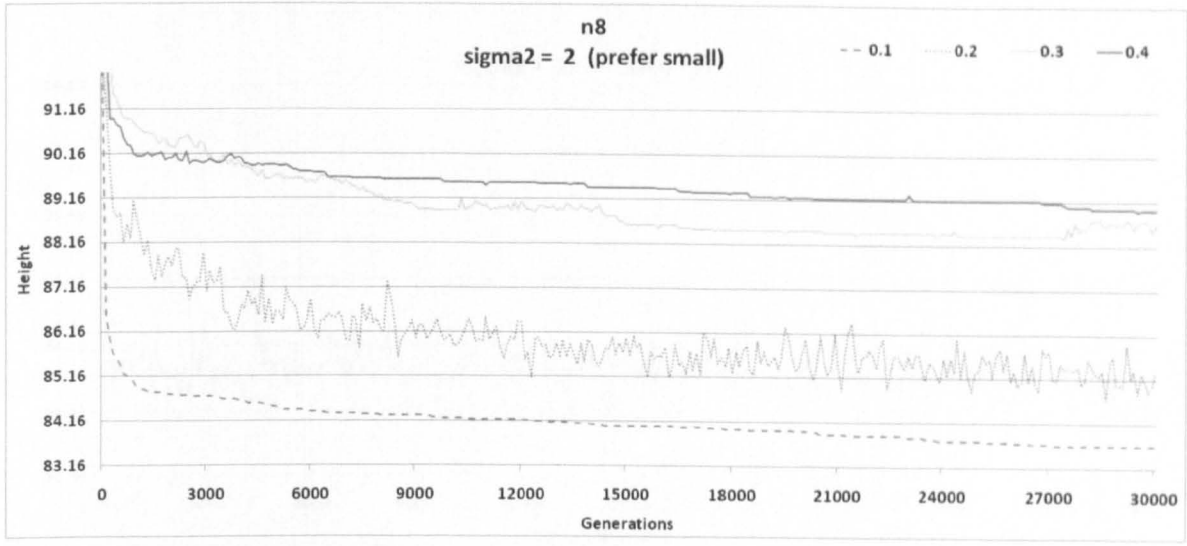


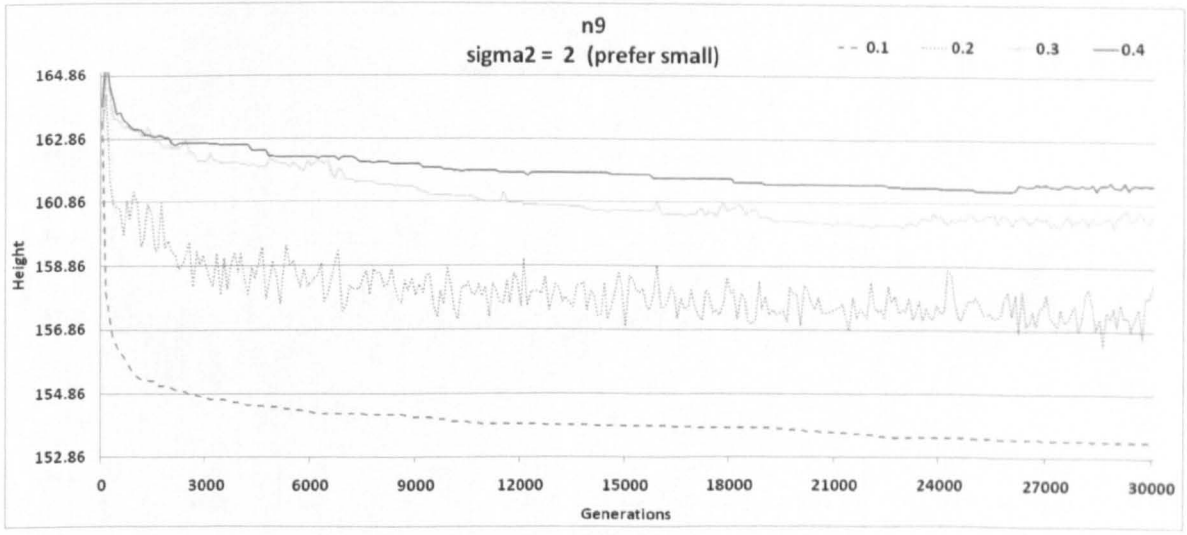


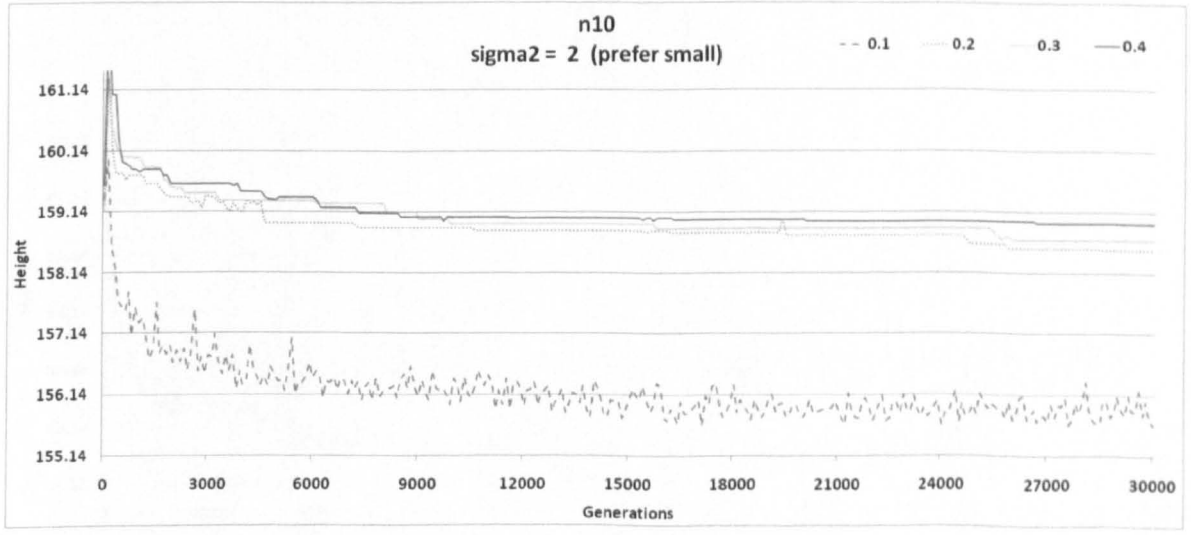

n5
sigma2 = 2 (prefer small)
0.1
0.2
0.3
0.4
110.46
109.46
108.46
107.46
106.46
105.46
104.46
103.46
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations


n6
sigma2 = 2 (prefer small)
0.1
0.2
0.3
0.4
111.34
109.34
107.34
105.34
103.34
101.34
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

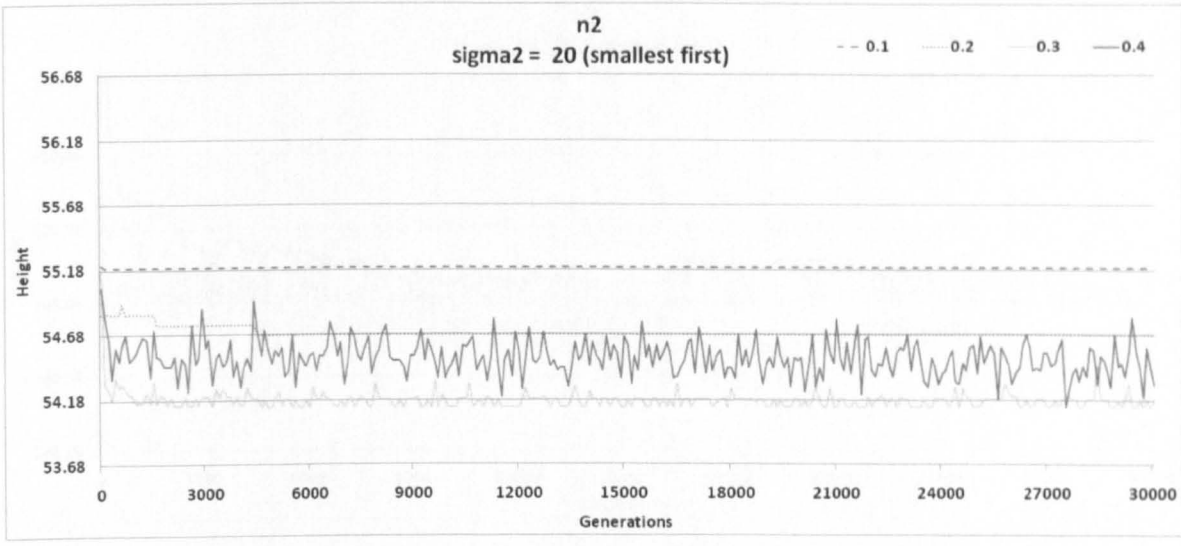

n7
sigma2 = 2 (prefer small)
0.1
0.2
0.3
0.4
118.44
116.44
114.44
112.44
110.44
108.44
106.44
104.44
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

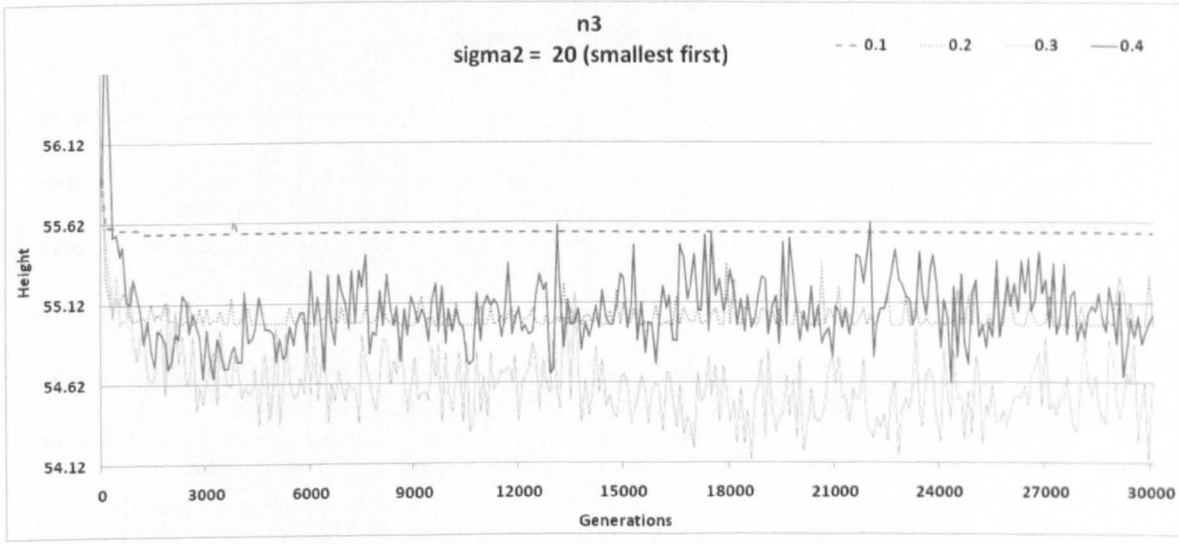

n8
sigma2 = 2 (prefer small)
0.1
0.2
0.3
0.4
Height
91.16
90.16
89.16
88.16
87.16
86.16
85.16
84.16
83.16
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

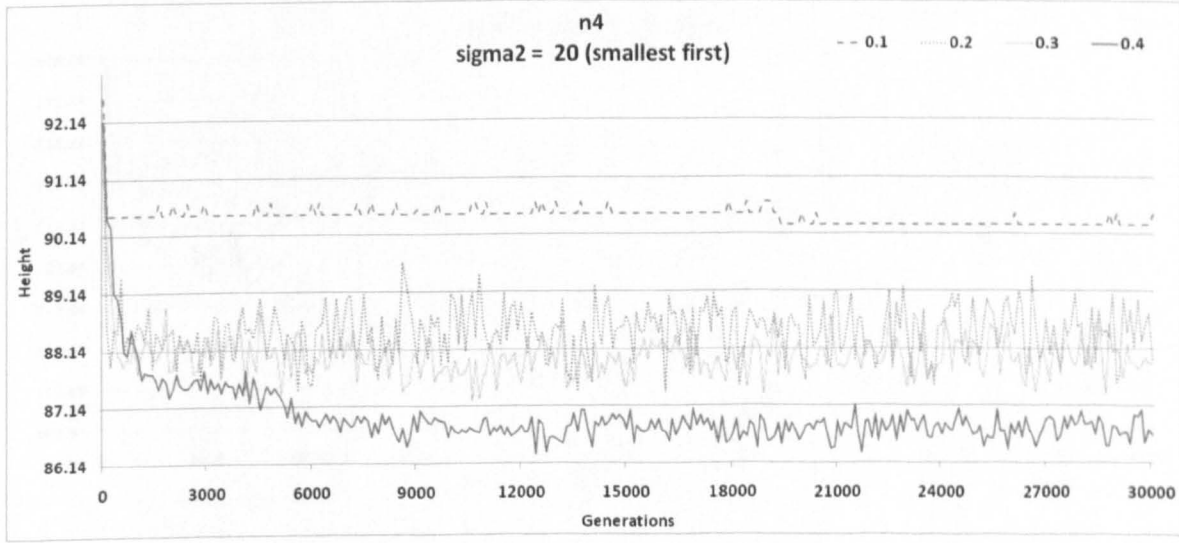

n9
sigma2 = 2 (prefer small)
0.1
0.2
0.3
0.4
Height
164.86
162.86
160.86
158.86
156.86
154.86
152.86
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

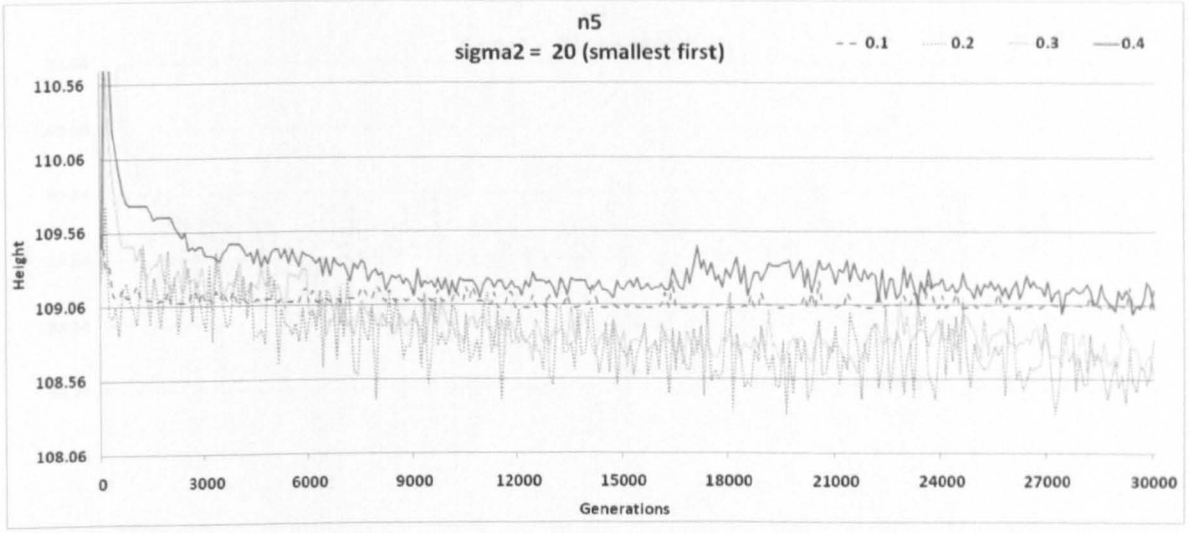

n10
sigma2 = 2 (prefer small)
0.1
0.2
0.3
0.4
Height
161.14
160.14
159.14
158.14
157.14
156.14
155.14
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

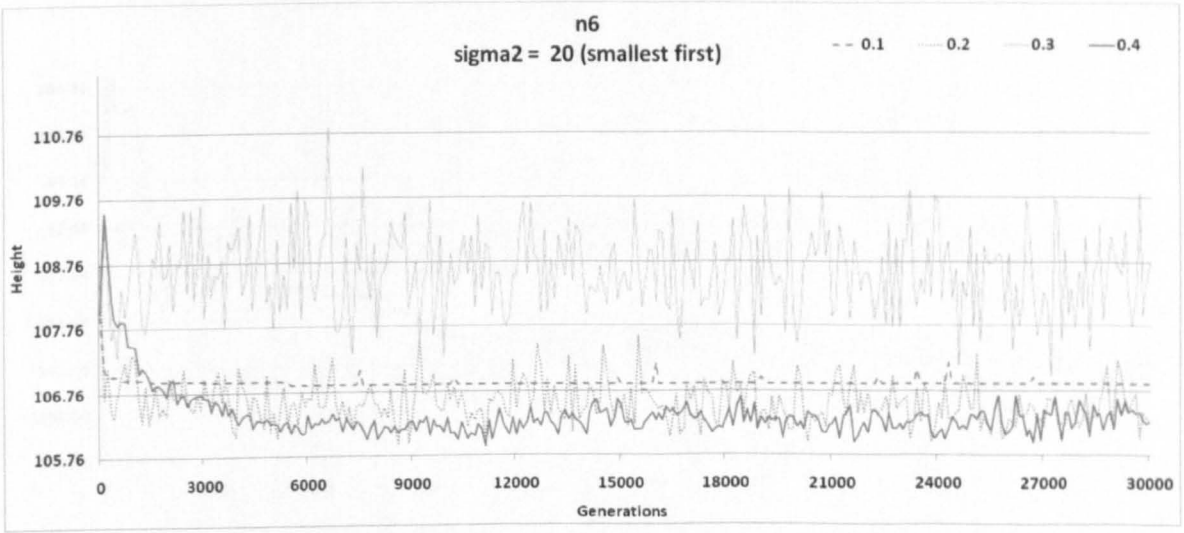
C.2.5 Smallest First

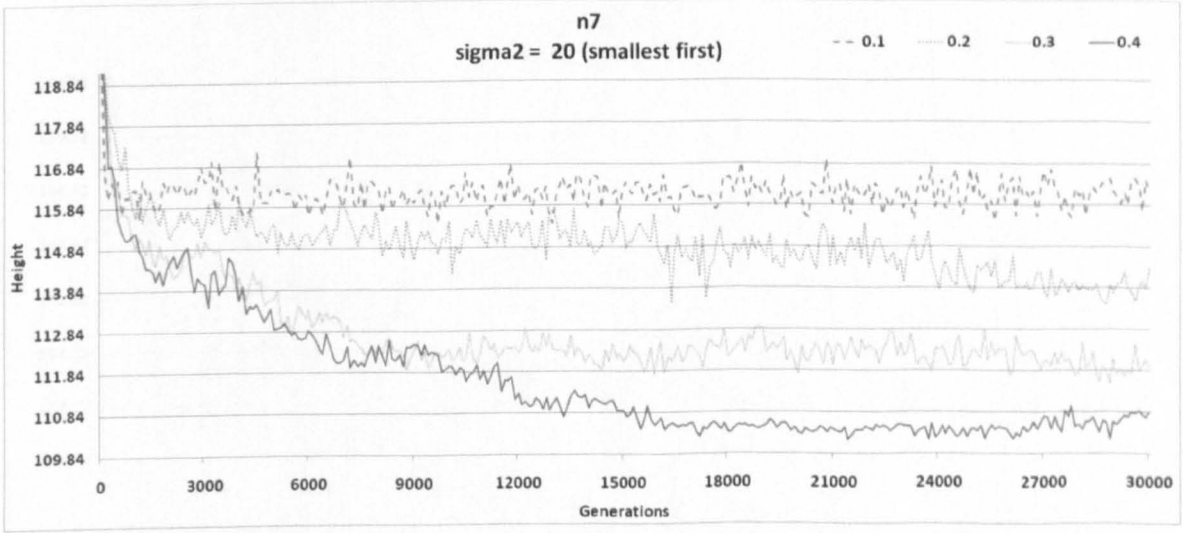


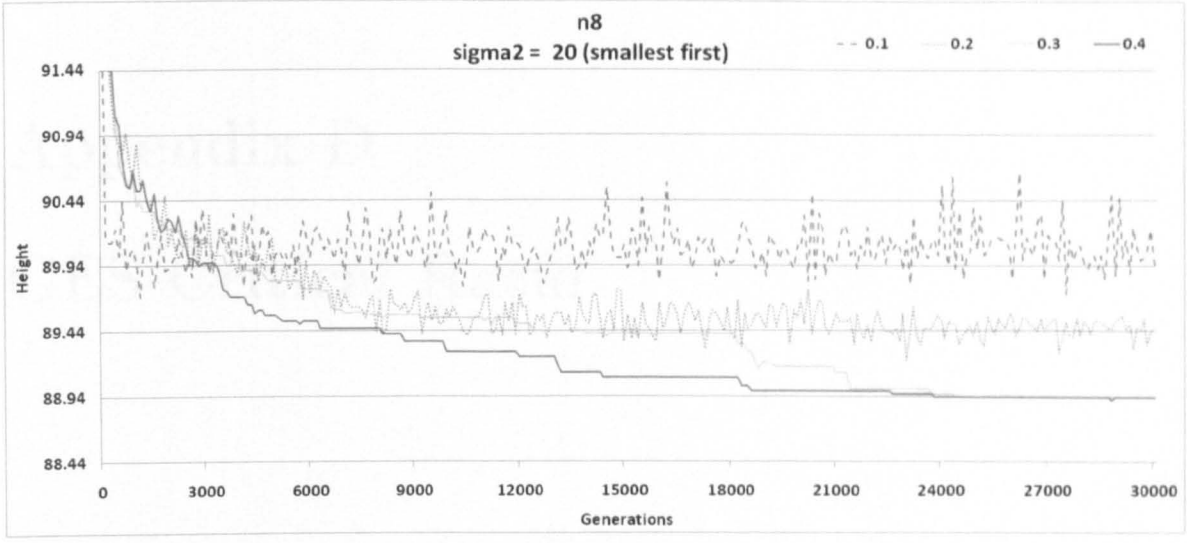


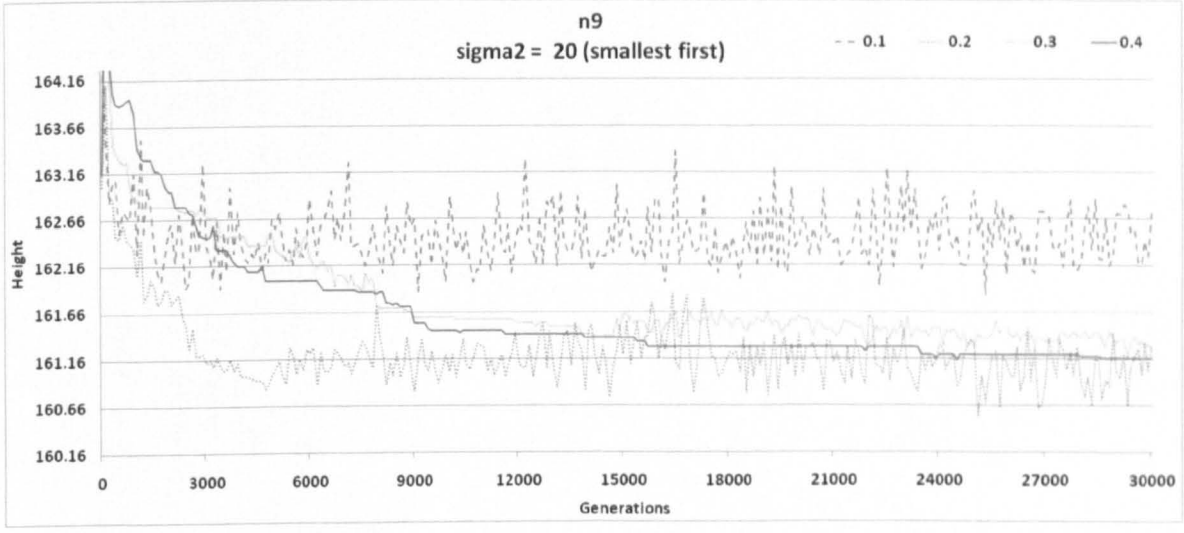


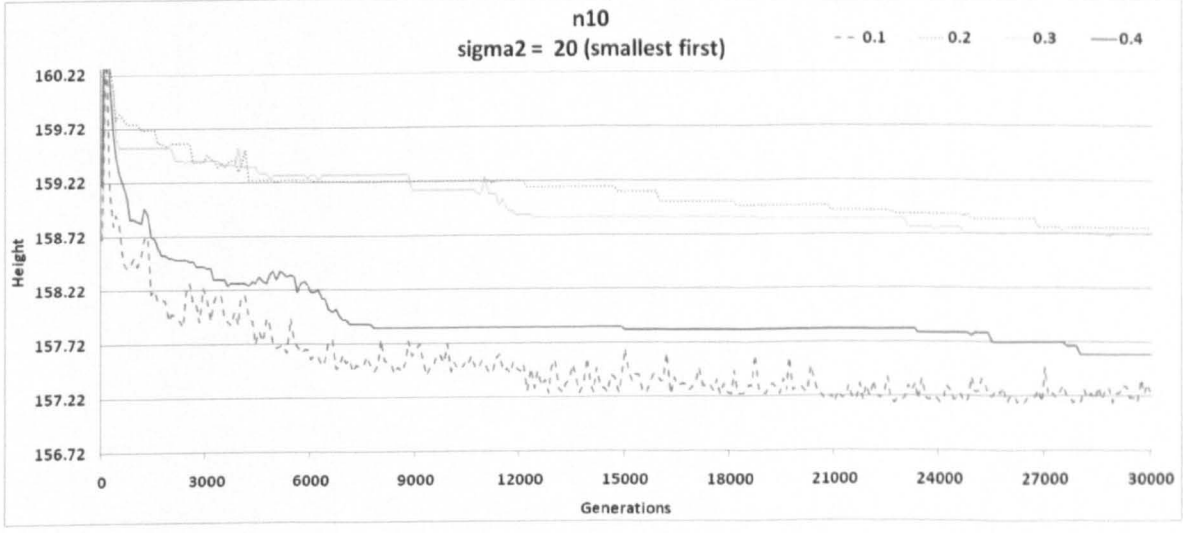

n5
sigma2 = 20 (smallest first)
0.1
0.2
0.3
0.4
110.56
110.06
109.56
109.06
108.56
108.06
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations


n6
sigma2 = 20 (smallest first)
0.1
0.2
0.3
0.4
110.76
109.76
108.76
107.76
106.76
105.76
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations


n7
sigma2 = 20 (smallest first)
0.1
0.2
0.3
0.4
118.84
117.84
116.84
115.84
114.84
113.84
112.84
111.84
110.84
109.84
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations


n8
sigma2 = 20 (smallest first)
0.1
0.2
0.3
0.4
91.44
90.94
90.44
89.94
89.44
88.94
88.44
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations


n9
sigma2 = 20 (smallest first)
0.1
0.2
0.3
0.4
164.16
163.66
163.16
162.66
162.16
161.66
161.16
160.66
160.16
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations


n10
sigma2 = 20 (smallest first)
0.1
0.2
0.3
0.4
160.22
159.72
159.22
158.72
158.22
157.72
157.22
156.72
Height
0
3000
6000
9000
12000
15000
18000
21000
24000
27000
30000
Generations

Appendix D

GES Critical Ratio

GES Critical Ratio (percentage of shapes in the Critical Group)

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
n1	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00
n2	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00
n3	51.67	51.90	51.32	51.46	51.04	50.96	50.69	50.76	50.75	50.75	50.96	51.15	51.30	51.42	51.38	51.43	51.42	51.78	51.16	51.30
n4	83.00	82.86	82.95	82.63	82.18	81.98	81.84	81.91	81.95	82.04	82.01	82.17	82.23	82.32	82.34	82.42	82.53	83.01	82.22	82.48
n5	104.43	104.25	104.09	104.26	103.82	104.03	103.77	103.70	103.68	103.72	103.84	103.92	104.01	104.21	104.24	104.23	104.38	105.20	104.15	104.25
n6	102.95	102.84	102.75	102.50	102.19	102.00	102.01	101.99	102.00	102.14	102.16	102.24	102.40	102.36	102.47	102.44	102.40	102.84	102.28	102.42
n7	105.20	103.45	102.38	102.53	102.82	103.28	103.55	103.79	103.88	103.89	104.17	104.18	104.24	104.29	104.41	104.55	104.31	104.72	103.76	103.84
n8	84.63	84.15	83.82	83.58	83.28	83.16	83.17	83.45	83.60	83.71	83.82	84.14	84.21	84.30	84.38	84.48	84.17	84.74	84.02	84.12
n9	154.29	153.66	153.63	153.69	153.55	153.80	153.82	153.84	154.05	154.21	154.45	154.60	154.64	154.60	154.52	154.63	154.81	154.97	154.57	154.58
n10	152.97	152.90	152.86	152.99	153.15	153.66	153.88	154.43	154.55	154.56	154.77	154.71	154.86	154.78	154.83	155.04	155.13	155.88	154.90	154.71

GES Critical Ratio (percentage of shapes in the Critical Group)

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
C1	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00
	20.57	20.52	20.97	20.98	20.92	20.86	20.86	20.91	20.83	20.94	20.98	20.97	20.97	20.95	20.99	20.97	20.95	21.00	20.90	20.93
	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00
C2	40.93	40.69	40.86	40.86	40.96	40.40	40.56	40.23	40.00	40.00	40	40.02	40.01	40.00	40.03	40.04	40.03	40.42	40.02	40.01
	40.97	40.96	40.98	40.90	40.97	40.95	40.69	40.66	40.69	40.56	40.72	40.75	40.81	40.80	40.79	40.73	40.77	40.98	40.65	40.76
	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00
C2t	15.68	15.86	15.08	15.07	15.11	15.38	15.34	15.39	15.34	15.45	15.4	15.72	15.87	15.79	15.88	15.88	15.92	15.82	15.78	15.90
	15.98	16.00	15.98	15.99	15.88	15.86	15.86	15.90	15.89	15.98	15.99	15.98	16.00	16.00	16.00	15.99	15.98	15.99	16.00	15.99
	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15	15.00	15.00	15.00	15.07	15.05	15.11	15.07	15.01	15.08
C3	62.63	62.44	62.24	62.54	62.37	62.16	61.47	61.24	61.33	61.37	60.92	61.17	61.25	61.20	61.43	61.37	61.32	62.28	61.19	61.24
	62.46	62.36	61.95	61.99	61.97	61.94	61.84	61.69	61.43	61.49	61.58	61.68	61.67	61.85	61.78	61.77	61.74	62.12	61.73	61.77
	61.63	61.57	61.48	60.70	60.68	60.92	61.04	60.52	60.59	60.72	60.64	60.59	60.90	61.05	61.17	61.20	61.03	61.81	60.97	61.09
C3t	31.00	30.99	30.97	30.98	30.93	30.95	30.93	30.96	30.89	30.94	30.99	30.96	30.96	30.97	31.00	31.00	31.00	30.99	31.00	30.98
	31.27	31.35	31.00	31.01	31.39	31.06	31.00	31.00	31.02	31.01	31.03	31.05	31.18	31.29	31.28	31.28	31.24	31.24	31.16	31.27
	31.05	31.06	31.00	31.00	31.01	30.85	30.89	30.76	30.82	30.87	30.93	30.91	30.96	30.98	31.00	30.98	31.00	30.97	30.97	30.99
C4	63.14	63.06	62.95	62.94	62.86	62.97	62.80	62.63	62.38	62.33	62.59	62.73	62.81	62.83	62.94	62.97	62.89	63.30	62.85	62.90
	62.92	62.89	62.83	62.55	62.35	62.51	62.05	62.07	62.27	62.39	62.52	62.73	62.83	62.83	62.91	62.98	62.98	63.21	62.89	62.79
	62.55	62.20	62.36	62.45	61.91	61.83	61.92	61.93	61.98	62.03	62.1	62.13	62.30	62.32	62.39	62.49	62.48	62.91	62.27	62.29
C5	63.01	62.96	62.97	62.94	62.92	62.85	62.27	62.34	62.55	62.74	62.99	62.97	63.04	62.98	63.06	63.08	63.15	63.63	63.09	63.07
	63.41	63.17	63.06	62.99	62.58	62.57	62.62	62.90	63.02	63.22	63.32	63.31	63.42	63.79	63.70	63.69	63.78	63.97	63.36	63.43
	63.00	63.06	62.97	63.00	62.84	62.83	62.67	62.57	62.57	62.78	62.95	62.93	63.08	63.12	63.16	63.28	63.36	63.72	63.16	63.28
C5t	94.28	94.14	94.06	93.93	93.64	93.68	93.23	93.18	93.18	93.31	93.54	93.59	93.68	93.82	93.91	94.06	94.16	94.08	93.98	93.99
	95.13	94.68	94.41	94.44	93.54	93.34	93.35	93.78	93.97	94.09	94.45	94.42	94.60	94.92	94.85	94.97	94.97	94.94	94.57	94.74
	94.45	94.49	94.46	94.27	94.08	93.93	93.65	93.30	93.26	93.72	93.72	93.88	94.08	94.31	94.32	94.51	94.57	94.51	94.15	94.30
C6	84.40	84.22	84.13	84.20	84.26	83.53	83.46	83.64	83.75	83.82	84.36	84.61	84.54	84.68	84.75	84.83	84.85	85.28	84.48	84.66
	85.04	84.83	84.13	83.53	83.20	83.82	84.13	84.34	84.47	84.84	84.94	84.95	85.03	85.17	85.21	85.34	85.20	85.85	85.09	85.11
	84.59	84.16	84.01	83.97	83.97	83.62	83.67	83.79	83.90	84.24	84.48	84.55	84.65	84.91	84.83	84.95	85.00	85.53	84.84	84.87
C6t	126.57	126.29	126.09	125.84	125.80	125.05	124.68	124.94	124.85	125.17	125.76	125.75	126.26	126.48	126.59	126.49	126.57	126.45	126.18	126.46
	126.09	125.71	125.41	124.86	124.04	124.49	124.94	125.16	125.39	125.71	126.08	126.16	126.58	126.62	126.75	126.74	126.80	126.95	126.36	126.46
	126.51	126.38	126.26	125.97	125.79	125.36	125.03	125.09	125.23	125.49	125.87	126.14	126.33	126.41	126.49	126.53	126.77	126.55	126.45	126.52
C7	170.95	170.82	170.57	170.38	169.85	169.10	168.82	169.02	169.20	169.68	170.45	170.84	170.99	171.76	171.58	171.84	171.87	173.40	171.36	171.64
	169.36	168.41	167.75	167.23	167.52	168.07	169.00	169.48	170.31	170.26	170.73	171.05	171.32	171.37	171.67	171.41	171.13	172.88	170.82	170.85
	171.52	170.80	170.38	169.06	168.58	168.24	168.61	169.54	169.57	170.69	171.22	171.62	171.99	172.15	172.44	172.37	172.56	173.94	171.85	172.14
C7t	254.63	254.22	253.96	253.57	252.97	252.97	252.74	252.10	252.02	252.63	253.52	254.48	254.56	255.37	255.51	255.66	255.64	255.88	255.08	255.46
	253.32	251.68	251.47	250.32	250.17	250.05	250.93	251.92	253.37	253.89	253.9	254.87	255.27	254.59	254.96	255.51	255.43	255.44	254.81	254.65
	254.17	252.99	252.34	251.49	251.21	251.17	251.18	251.90	252.47	253.97	254.06	254.41	254.62	255.17	255.07	255.30	255.56	255.31	254.88	254.97

GES Critical Ratio (percentage of shapes in the Critical Group)

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
N1	210.88	210.98	212.04	212.78	208.57	207.86	208.31	207.93	207.87	208.11	208.52	208.20	208.07	207.50	208.90	207.82	208.72	208.62	207.63	208.41
	210.99	212.46	212.13	208.09	205.50	205.59	205.98	206.22	206.85	205.66	206.85	206.77	207.01	206.85	207.59	207.45	207.38	207.35	206.73	207.67
	210.79	210.65	210.76	210.36	205.41	204.24	203.46	206.93	205.27	205.86	208.33	206.94	208.54	208.42	208.46	209.27	208.29	208.37	208.51	209.12
	210.96	209.79	210.34	205.77	207.38	206.14	205.69	206.18	205.50	207.62	207.26	207.94	208.12	208.19	208.26	207.44	206.68	207.13	206.16	207.27
	208.61	210.36	209.96	209.59	209.31	209.40	210.21	208.86	208.33	208.57	208.28	209.08	208.63	208.68	208.79	209.12	208.86	209.16	208.51	208.65
N2	212.41	212.46	212.81	211.47	211.77	211.92	210.84	210.71	210.76	210.90	210.88	210.87	211.12	211.05	211.23	210.94	211.05	210.83	210.60	210.70
	212.87	212.58	210.63	210.97	210.78	211.42	209.35	209.38	208.58	208.59	208.89	208.86	208.94	208.77	209.12	209.17	209.14	209.20	208.63	208.83
	213.80	212.69	212.62	212.80	213.14	212.73	213.31	212.16	211.42	211.33	211.25	211.68	211.77	211.28	211.76	211.80	212.05	212.01	211.48	211.77
	213.67	213.74	213.37	212.23	213.58	213.36	212.38	210.96	210.48	210.80	210.75	210.87	211.33	211.17	211.35	211.16	211.47	211.60	211.10	211.48
	214.14	214.30	212.37	212.06	211.77	212.43	212.54	212.22	209.85	210.07	210.2	210.33	210.41	210.59	210.64	210.59	210.65	210.47	210.26	210.51
N3	213.56	213.22	212.69	212.90	212.35	211.92	211.52	211.41	211.20	211.15	211.68	211.70	211.91	211.82	211.87	211.73	211.93	212.18	211.95	211.83
	213.98	214.42	213.45	213.10	211.30	211.50	210.84	210.45	210.98	211.17	211.17	211.57	211.76	212.08	212.40	212.12	211.98	212.12	211.65	212.08
	211.22	211.94	211.58	211.24	211.05	211.22	210.10	209.81	209.66	209.76	210.36	210.40	210.55	210.83	210.70	210.49	210.21	210.22	210.06	210.06
	216.07	214.04	213.24	212.76	213.75	212.80	211.85	211.57	211.36	211.44	211.3	211.80	211.66	211.82	211.88	211.65	211.72	211.36	211.03	211.64
	213.09	212.21	211.65	211.10	211.48	211.45	210.95	210.54	210.58	210.48	210.29	210.53	210.72	210.59	210.66	210.61	210.97	211.05	210.62	211.03
N4	214.37	213.61	213.52	212.51	212.54	212.60	211.96	211.48	211.00	211.07	211.51	211.76	212.06	211.76	212.29	212.90	212.55	212.39	212.42	212.67
	215.04	214.68	214.49	214.91	211.61	211.12	210.46	210.66	211.07	211.77	212.31	212.33	212.43	212.67	212.98	213.03	213.03	213.16	212.81	212.61
	212.33	212.35	211.37	211.45	211.53	211.69	211.31	210.83	210.57	210.09	210.17	210.97	211.10	211.29	211.31	211.46	211.45	211.66	211.23	211.49
	212.93	212.54	211.41	210.69	211.10	210.90	210.76	210.46	210.00	210.01	210.87	210.62	210.68	211.78	211.44	211.74	211.78	212.02	211.14	211.38
	213.64	213.21	212.01	212.09	212.23	211.47	209.78	210.16	210.20	210.60	210.8	211.49	211.55	212.33	212.27	212.63	212.22	212.05	212.04	211.86
N5	213.78	213.25	213.22	213.17	213.04	212.78	211.00	210.99	211.92	211.77	212.33	212.76	213.22	213.53	213.65	213.30	213.71	213.52	212.93	213.24
	213.53	212.66	211.98	212.11	211.81	211.68	211.25	210.55	210.68	210.81	211.85	212.04	212.01	212.30	212.56	212.88	212.82	212.44	212.33	212.36
	214.16	213.01	212.64	212.24	211.60	210.67	211.20	211.50	211.93	212.87	212.91	213.19	213.65	213.97	213.99	214.19	214.09	214.78	213.91	213.92
	216.27	216.00	215.70	215.40	215.11	215.41	213.91	213.26	212.80	213.19	213.43	213.38	213.76	213.95	214.59	214.79	215.02	215.58	214.66	214.24
	213.61	213.01	212.47	211.86	212.12	211.98	211.12	211.45	211.78	212.36	212.46	213.05	213.22	213.88	213.37	213.40	213.20	214.67	213.17	213.21
N6	213.29	212.88	212.24	212.13	211.27	210.93	210.23	210.83	211.28	211.68	212.38	212.69	213.55	213.08	213.57	213.59	213.20	214.81	212.80	212.88
	213.13	211.40	211.14	210.44	209.94	209.80	209.94	210.44	211.09	211.81	212.5	213.08	213.22	212.95	213.34	213.37	213.19	215.06	212.90	213.12
	213.05	212.10	211.45	211.28	210.08	209.97	209.47	210.00	210.18	211.45	212.17	212.84	212.95	213.00	213.65	213.57	213.75	214.59	212.72	213.00
	214.60	214.02	213.75	213.68	213.66	212.57	211.66	211.61	212.12	212.46	213.18	213.44	213.59	213.98	214.29	214.42	214.37	215.73	213.99	214.11
	212.11	211.42	210.95	211.38	209.96	209.82	210.37	210.68	211.45	211.76	212.44	212.69	212.96	213.02	213.20	213.27	213.27	214.47	212.90	213.02
N7	212.11	211.31	210.90	209.33	208.46	208.71	209.43	210.57	211.41	211.54	212.47	212.99	213.39	213.55	213.63	213.75	213.47	215.87	213.27	213.01
	211.38	210.59	209.96	209.11	208.70	209.19	210.18	211.04	211.45	212.12	212.6	212.53	213.10	213.61	213.04	213.62	213.82	215.62	213.01	213.17
	211.74	211.30	210.13	209.94	209.07	209.05	209.86	211.16	211.36	212.24	212.79	213.11	213.80	213.52	213.69	213.87	213.97	215.15	212.89	213.14
	213.95	212.88	212.25	210.98	209.91	210.12	210.34	210.98	212.39	212.91	213.41	214.08	214.57	214.77	214.67	214.93	214.92	216.50	214.09	214.31
	211.21	210.41	210.17	210.16	209.60	209.16	209.61	210.01	210.75	211.66	212.32	212.76	213.12	212.88	212.93	213.32	213.35	214.53	212.50	212.87

GES Critical Ratio (percentage of shapes in the Critical Group)

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
T1	209.25	207.00	200.11	206.13	201.54	202.02	201.46	202.11	201.95	202.25	203.14	204.36	204.29	204.02	206.30	206.00	205.50	205.48	204.91	205.46
	210.58	209.70	210.49	206.46	206.32	205.61	204.92	205.90	205.98	207.61	208.08	207.57	207.96	208.07	208.56	207.77	207.00	206.88	206.85	207.27
	201.13	200.14	200.12	200.00	200.83	201.07	200.50	200.52	200.03	200.05	200.04	200.31	200.49	200.54	200.94	200.86	200.09	200.41	200.00	200.41
	205.76	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.16	201.11	201.73	201.80	201.49	202.17	202.71	202.19	201.55	202.10
	210.42	212.56	211.27	211.02	204.17	202.25	201.17	200.66	202.78	202.52	204.39	204.98	205.77	204.80	204.74	204.49	205.47	205.53	204.54	206.12
T2	209.74	209.25	209.04	208.94	208.82	208.28	208.08	207.91	208.05	207.82	208.53	208.33	209.23	209.08	209.17	208.99	208.76	208.93	208.61	208.97
	212.48	212.27	211.31	211.63	211.55	210.78	211.10	210.87	210.53	210.75	211.05	211.56	211.71	211.87	211.69	211.38	211.18	211.17	211.14	211.36
	213.58	213.95	214.31	213.51	212.84	211.58	210.63	210.60	210.85	210.95	210.88	210.97	211.20	212.00	211.80	211.76	211.62	211.31	210.93	211.14
	213.92	214.34	212.58	212.27	212.16	212.47	212.53	212.18	210.27	210.22	210.65	210.42	210.10	210.47	210.22	210.58	210.41	210.50	210.39	210.50
	213.08	212.26	210.52	210.98	210.97	211.15	209.61	209.43	208.92	208.73	208.78	208.66	208.65	208.67	209.15	209.05	209.32	209.18	208.74	209.16
T3	215.64	214.34	213.24	212.88	213.72	213.21	211.97	211.51	211.19	211.11	211.42	211.64	211.89	211.60	211.89	211.74	211.43	211.43	211.10	211.49
	214.13	213.63	212.99	212.89	212.45	211.51	211.37	211.24	211.27	211.25	211.39	212.02	211.98	212.10	212.25	212.14	212.55	212.40	211.86	212.03
	212.98	212.26	212.15	211.94	213.23	212.15	212.24	212.14	210.02	210.16	210.29	210.32	210.43	210.72	210.87	210.39	210.90	210.79	210.49	210.65
	212.49	211.77	210.42	210.05	210.79	208.64	208.46	207.95	207.69	207.56	207.83	207.94	208.28	208.48	209.07	209.13	208.93	208.94	208.53	208.82
	214.01	214.14	213.86	213.20	211.35	211.27	210.78	210.73	210.80	211.05	211.15	211.81	211.91	211.93	211.97	211.96	212.13	212.03	211.49	212.03
T4	212.49	211.90	211.75	211.68	211.77	212.29	212.74	210.54	210.31	210.71	211.21	211.45	211.22	211.74	211.86	212.22	212.33	211.86	211.54	211.85
	212.30	212.34	211.56	211.46	211.46	211.57	211.66	210.81	210.59	210.37	210.42	210.91	211.06	211.35	211.41	211.47	211.36	211.55	211.30	211.50
	213.02	212.55	211.21	210.86	210.78	211.03	210.66	210.53	209.64	209.97	210.55	210.76	210.70	211.33	211.34	211.66	211.85	211.87	211.24	211.47
	214.23	213.57	213.41	212.44	212.53	212.47	212.04	211.54	210.98	210.91	211.29	211.67	212.09	212.05	212.39	212.76	212.97	212.62	212.53	212.75
	215.18	214.69	214.33	214.79	211.92	211.17	210.69	210.71	211.07	211.59	212.03	212.50	212.51	212.71	212.84	212.84	213.12	213.10	212.82	212.74
T5	214.30	212.92	212.62	212.27	211.60	210.85	211.12	211.51	212.07	212.42	213.1	213.63	213.66	213.79	214.45	214.17	214.23	214.24	213.91	213.60
	212.95	212.07	211.84	211.64	210.84	211.08	209.90	210.11	211.08	211.58	212.23	212.49	212.53	212.78	212.71	213.32	213.58	212.93	212.82	212.83
	214.04	213.24	213.16	213.04	213.33	212.90	210.95	211.34	211.50	211.60	212.36	212.75	213.15	213.30	213.68	213.55	213.53	213.49	212.77	213.16
	216.24	215.86	215.73	215.33	215.14	215.45	213.80	213.58	212.78	213.24	213.26	213.63	213.63	214.17	214.37	214.93	214.72	214.74	214.34	214.44
	213.45	212.57	211.97	212.06	211.78	211.84	211.05	210.33	210.49	210.96	211.79	212.27	212.13	212.17	212.51	212.56	212.52	212.69	212.40	212.63
T6	215.59	214.82	214.24	213.17	212.46	210.80	209.95	211.05	211.87	211.93	212.5	213.07	213.23	213.55	214.06	213.94	214.18	214.75	213.60	213.89
	212.95	212.02	211.38	211.33	209.98	210.12	209.75	209.87	210.31	211.59	212.11	212.68	213.24	213.36	213.64	213.74	213.50	213.65	212.68	212.84
	212.20	211.45	211.01	211.40	210.04	209.92	210.22	210.64	211.05	211.68	212.48	212.56	212.98	213.19	213.61	213.20	213.40	213.49	212.79	213.11
	213.20	211.47	210.89	210.31	209.96	210.00	209.84	210.41	211.03	212.04	212.56	212.84	213.12	213.18	213.13	213.69	213.28	212.92	212.81	213.23
	211.84	211.09	210.81	210.58	210.61	209.47	210.12	210.40	211.40	211.44	211.99	212.47	212.92	213.00	213.12	213.21	212.90	213.11	212.52	212.78
T7	212.19	211.39	211.38	210.96	210.85	209.86	209.46	210.17	211.03	211.83	212.75	213.09	213.49	213.93	213.63	213.63	214.19	213.98	213.21	213.70
	213.07	212.41	211.93	211.65	211.03	210.03	210.01	210.39	211.15	212.15	213.17	213.09	213.95	214.18	213.99	214.29	214.40	213.77	213.52	213.84
	213.69	212.92	212.74	212.87	212.71	211.17	210.88	211.16	211.97	212.43	213.24	213.70	213.97	214.85	215.01	214.75	214.80	214.93	214.05	214.56
	211.62	211.03	210.73	210.66	209.93	209.53	209.40	209.18	209.86	210.34	211.31	212.18	212.21	212.71	212.84	212.95	213.33	212.99	212.27	212.67
	213.30	213.17	212.90	212.65	212.64	211.43	210.50	211.17	211.75	212.36	212.96	213.44	213.75	214.00	214.65	214.36	215.07	214.86	214.13	214.48