Formulation Space Search for two-dimensional Packing Problems

a thesis submitted to

Brunel University

in the subject of Operational Research

for the degree

of Doctor of Philosophy

by

Claudia Orquídea López Soto

March 20, 2013

Abstract

The two-dimension packing problem is concerned with the arrangement of items without overlaps inside a container. In particular we have considered the case when the items are circular objects, some of the general examples that can be found in the industry are related with packing, storing and transportation of circular objects. Although there are several approaches we want to investigate the use of formulation space search. Formulation space search is a fairly recent method that provides an easy way to escape from local optima for non-linear problems allowing to achieve better results. Despite the fact that it has been implemented to solve the packing problem with identical circles, we present an improved implementation of the formulation space search that gives better results for the case of identical and non-identical circles, also considering that they are packed inside different shaped containers, for which we provide the needed modifications for an appropriate implementation. The containers considered are: the unit circle, the unit square, two rectangles with different dimension (length 5, width 1 and length 10 width 1), a right-isosceles triangle, a semicircle and a right-circular quadrant. Results from the tests conducted shown several improvements over the best previously known for the case of identical circles inside three different containers: a right-isosceles triangle, a semicircle and a circular quadrant. In order to extend the scope of the formulation space search approach we used it to solve mixed-integer non-linear problems, in particular those with zero-one variables. Our findings suggest that our implementation provides a competitive way to solve these kind of problems.

Acknowledgements

I would like to express my profound gratitude to my supervisor Professor John Beasley for his patience guidance, invaluable support, but most of all for his time during this research.

I wish to acknowledge the help and support provided by the administrative and technical staff of the mathematics department at Brunel University, specially to Kay, Linda, Frances, Neil and Nalin.

I would like to thank to the Mexican National Council for Science and Technology (CONACyT) for their financial support.

I would like to give my thanks to my friends and family with special mention to my grand-mother for all the support and understanding.

Last, but not least, I would like to extend my special thanks to my husband Carlos E. Rodriguez for his patience, encouragement on the completion of this work and for making of this a more enjoyable experience.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In recent years, there has been an increasing interest in the circle packing problem, several algorithms have been developed in order to achieve the best solution. Although it is a very well known and studied problem we want to investigate if improvements can be made if we address the problem from a different perspective like formulation space search with additional strategies. Formulation space search is a recent method that proposed an easy and interesting way to overcome the difficulties found in non-linear optimisation problems of being unable to leave local optima. We also want to extend this approach to more complicated problems such as mixed-integer non-linear programming problems. The outline of this dissertation is as below.

Chapter 2 offers a short but comprehensive introductory description of the packing problem along with a literature survey of the most recent work related to the circle packing problem addressing different variants: identical circles, non-identical circles and the different containers that have been considered so far in the literature. It also presents the literature found concerning formulation space search and a couple of its applications.

Chapter 3 constitutes the general framework that provides us with the essential tools to draw new paths to follow in this research. It presents the design of the heuristic algorithm implemented for the circle packing problem with identical circles inside a

circular container which is considered as the basic instances to address. It also presents the several experiments conducted in order to investigate how different modification could affect the behaviour of the strategy adopted.

Chapter 4 shows the immediate extension of our findings in Chapter 3, it describes in detail the appropriate modifications for the implementation of our formulation space search to different shaped containers. The numerical results achieved for some containers are improvements over the previous best-known solutions.

Chapter 5 describes the new scaled formulation model adopted to address the packing problem with non-identical circles inside a circular container using a close related approach to formulation space search called reformulation descent. It details the procedures adopted to reduce the size of the problem aiming to achieve good quality solutions and reducing the computational time.

Chapter 6 extends the scaled formulation of the packing problem presented in the previous chapter detailing the modifications needed to implement it for different shaped containers. We present new results for the instances considered.

Chapter 7 presents our findings concerning two approaches considered to solve mixed-integer non-linear zero-one problems using our formulation space search approach.

Chapter 8 presents the conclusion and contributions derived from the research carried out, it also presents a section describing possible avenues for future work.

It is worth noting that in Appendix A we present our findings concerning a particular approach for the packing problem with non-identical circles with just a few successful results.

Chapter 2

Literature Review

This literature review has the intention to give a general idea of the methods that have been used to tackle the circle packing problem in recent years. Let us say now that Castillo et al. [14] and Hifi et al. [33] present a comprehensive review of the packing problem considering circular and rectangular containers. In [14] the authors present a general model for the specific case discussed providing their respective industrial application. They give solutions from their numerical experiments conducted with different global optimizer software packages. In [33] the authors give a detailed review of what they called efficient approaches for the packing problem in two and three dimensions.

In the following sections we will highlight the work that to our consideration is relevant to our research: the circle packing problem and formulation space search. There are many different ways to approach the packing problem: by method used, by the size of the circles being identical or not, by the shape of the container, etc. Hence we have decided to divide this chapter into six sections: In Section 2.1 we give a general description of the packing problem with some examples. In Sections 2.3 and 2.4 we review works that present an approach to solve the circle packing problem for identical and non-identical circles respectively. As for Section 2.5 we present those works that use the formulation space search. In Section 2.2 we give a brief historical note and finally in

Section 2.6 we conclude this chapter with a summary.

2.1 Circle packing problems

In this section we give a general description of the two-dimensional packing problem presenting a few pictures generated with results from the heuristic developed in the present work to illustrate what the problem is about.

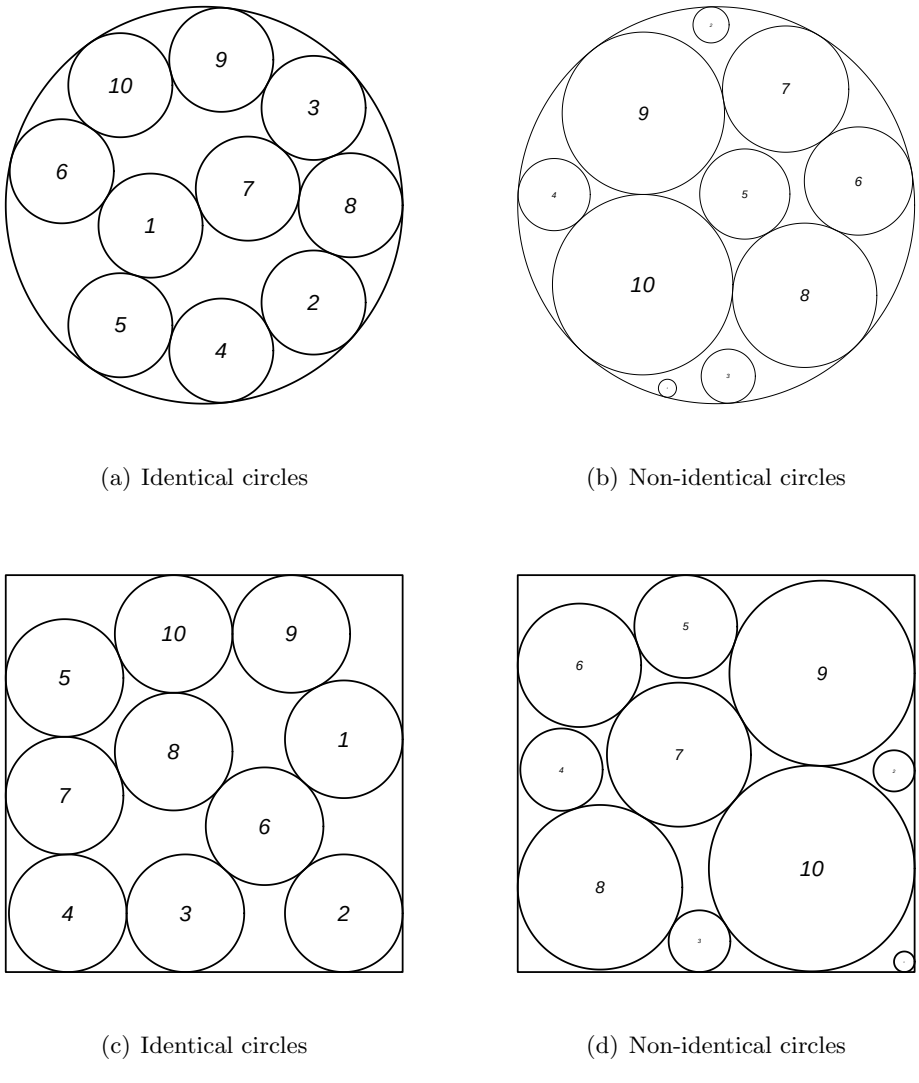
The circle packing problem is concerned with the arrangement of a finite number (denoted as n) of circles inside a container without overlaps. The mathematical model that represents the circle packing problem consists mainly of two types of constraints: the container boundary and the non-overlapping constraints. The container boundary constraints ensure that all circles lie inside the container whilst the non-overlapping constraints ensure that for any two given circles, the distance between their centres are at least the sum of their radii. When both sets of constraints are satisfied we have a feasible solution for the packing problem.

But in searching for an "optimal" solution that has one of the following two interpretations: Maximise the radii of all circles or to minimise the size of the container. These are two different but equivalent points of view in which the packing problem can be tackled, to exemplify let us consider the circular container.

For the first point of view the problem deals with maximising the radii of n circles with a fixed size container, in this case the unit circle (a circle of radius 1 with centre at the origin of the Cartesian plane), we denote as ρ_{max} the maximum radii obtained for all n circles to be packed without overlaps. However for the second point of view we want to pack n unitary circles (circles of radii 1) without overlaps inside a circular container with minimum radius $1/\rho_{max}$. Some work is focused on minimising the radius of the container whilst others in maximising the radii of the circles as we will note in the following sections.

Regarding the size of the circles to be packed the problem is categorized as that of

"packing identical circles" when all n circles have the same radius or as that of "packing non-identical circles" when the n circles to be packed do not have the same radius. Figure 2.1 illustrates four examples of the packing problem, in Figure 2.1(a) we have 10 identical circles inside a circular container, if we observe Figure 2.1(b) we immediately know that it is representing 10 non-identical circles inside a circular container. Figure 2.1(c) and Figure 2.1(d) depict the identical and non-identical packing of 10 circles inside a square container. Even though the circular and square containers are the most studied in the literature in our work we address these and other containers.



(a) Identical circles

(b) Non-identical circles

(c) Identical circles

(d) Non-identical circles

Figure 2.1: Packing $n = 10$ circles inside a circular and a square container

The most recent improvements for solutions to packing problems are reported at the website Packomania [72] that started in 1999 reporting the best known solutions for the packing problem inside a square container considering $n = 1$ to 200 identical circles. Since then the instances have increased not only in number, the containers under consideration have been diversified. For packing identical circles the containers are: circle, rectangles, right-angle isosceles triangle, semicircle, the right quadrant of a circle, as for the case of packing non-identical circles the containers are only circular. This website [72] is continually updated with results from all work addressing the packing problem in its different versions or approaches. It is also updated with results to a continuously running search overseen by the website owner.

2.2 A historical note

Evidence of first glances of interest in the circle packing problem are presented in the first chapter of the book by Szabo et al. [76], the authors give a brief historical review presenting among others Malfatti's problem (1803) and the context under which Farkas Bolyai (1775-1856) approached the packing problem.

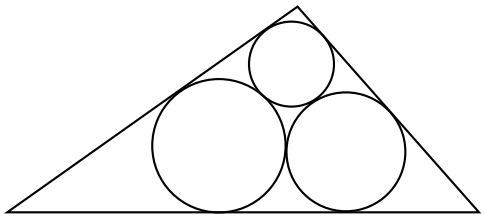


Figure 2.2: The Malfatti circles

An Italian mathematician called Gian Francesco Malfatti (1731-1807) proposed a solution to a problem where it was required to extract three cylindrical columns (not necessarily equal) from a piece of marble that consisted of a right triangular prism shape achieving minimum waste of the marble prism. This problem can be interpreted as

a two-dimensional problem of maximising the area/perimeter of the circles (whether identical or not) inside a right-angle triangle. A graphical representation of the solution given by Malfatti is illustrated in Figure 2.2. Even though this is known as Malfatti's problem there is evidence that it was earlier approached by the Japanese mathematician Chokuen Ajima (1732-1798) [76]. Moreover, in 1930 Lob and Richmond showed that Malfatti's solution is not the densest, later on Howard Eves in 1946 proposed another configuration improving the one of Malfatti's [7]. Finally Michael Goldberg in 1967 showed that Malfatti's configurations are never the solution [7, 76].

According to [76] when a Hungarian mathematician called Farkas Bolyai (1775-1856) was about to apply for a position for the forestry commission he was commissioned to study problems such as "planting trees in given regions such that they share the same amount of light and air". This practical application can be interpreted as the packing problem where the trees are represented by the centres of the circles inside a given bounded region.

In 1771 Joseph Louis Lagrange proved that the densest arrangements of identical circles in the plane was given by a hexagonal pattern that resembles a honeycomb, where the vertices of each hexagon represent the centres of a circle as depicted in Figure 2.3. Later on the Norwegian mathematician Axel Thue (1863-1922) provided the first optimal solution to the problem, although his proof was not very convincing. The Hungarian mathematician László Fejes Tóth (1915-2005) completed the work in 1940 proving its optimality. It is worth mentioning that this problem is the two-dimensional version of the known "Kepler conjecture". According to [7] in 1591 Sir Walter Raleigh wanted to know how to best stack cannonballs in his ship, Thomas Harriot who was his friend accomplished the task. Later on, Harriot convinced Kepler to adopt an atomic theory in his studies to describe the origin of the hexagonal shape of snowflakes (assuming that the snowflakes were composed of tiny spheres). By 1611 Johannes Kepler states the conjecture: "It says that no arrangement of equally sized spheres filling space has

a greater average density than that of the cubic close packing (face-centred cubic) and hexagonal close packing arrangements". Several attempts to prove the conjecture have been made and in [7] we can find a reproduction of an email sent by Thomas Hales to his colleagues on Sunday 9th of August 1998 announcing and distributing a copy of the solution to the Kepler conjecture, stating that the solution consists of 250 pages, plus the computer files of code and data files for combinatorics, interval arithmetic and linear programs. After four years under revision the referees were 99% confident that the solution is correct. In [7] we can find an extensive collection of examples and anecdotes where the idea of the packing problem play a vital role. The examples vary from different fields such as mathematics, physics, chemistry and technology.

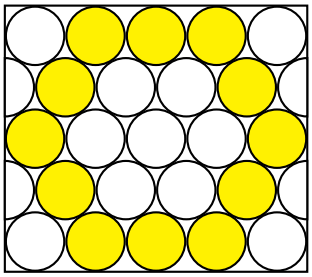


Figure 2.3: Densest pattern for identical circles in the plane

2.3 Packing identical circles

This section focuses on relatively recent literature that tackles the problem of packing identical circles. It is divided into two sections according to the shape of the container. Section 2.3.1 presents work dedicated to the circular container whilst Section 2.3.2 considers work relating to rectangular containers. Finally Section 2.3.3 deals with other containers.

2.3.1 Circular container

In recent years several approaches have been developed regarding the packing of identical circles inside a circular container. Birgin et al. [10] implemented a twice differentiable non-linear model of the packing problem for identical and non-identical circles using a non-linear solver called ALGENCAN. The strategy to reduce the number of the non-overlapping constraints starts by drawing a grid over the container. The algorithm will not consider the non-overlapping constraints of any two circles if their centres lay in non-adjacent squares of the grid. In other words, the non-overlapping constraints of any two circles are solely considered by the algorithm if their centres lay in adjacent squares of the grid, the rest are disregarded. They consider the packing of identical and non-identical circles in different shape containers such as: a circle, a square, a strip, a rectangle, an equilateral triangle and their equivalent in 3-D space: a sphere, a cube, a strip, a cuboid, a tetrahedron, a pyramid and a cylinder. Although the computational results presented are up to 100 circles there was a lack of accuracy in claiming to have beaten the best known results (as noted in [9]).

Afterwards, Birgin et al. [9] extended previous work [10] focusing solely on the case of identical (unitary) circles. They introduce a correction step after a feasible solution given by the non-linear solver ALGENCAN. For the correction step circles called "rattlers" are removed. A circle denominated as a rattler is a circle that is not touching any other circle nor the container. Then with the elements of the subset that contains the remaining circles they form an active non-linear system of equations, that is, the non-linear system of equations will be formed only by the constraints of the circles that are tangent to the container and by the constraints of the circles that are tangent to each other. The non-linear system is solved by the Newton-Raphson method, after convergence the rattlers are reintroduced. They give computational results for up to 50 unitary circles, for the circular and square container they match the best known results and present results for a rectangle and a strip container.

Grosso et al. [23] propose an algorithm based on Monotonic Basin Hopping (MBH) [45, 78] along with its variant, population basin hopping [24] to solve the circle packing problem of identical and non-identical circles inside a circular container with minimum radius. Monotonic basin hopping is a single solution iterative local search method with a perturbation move that allows the diversification of the solution by jumping from one local minimum to a closer one. Monotonic Basin Hopping arose as a variant of the Basin Hopping algorithm presented in [78], the main difference between both algorithms relies on the acceptance criteria of a given solution, whilst Basin Hopping accepts a worse solution with the help of a distribution probability, Monotonic Basin Hopping only accepts improved solutions. With regards to the Population Basin Hopping algorithm, it is a general case of MBH mentioned above, it uses a perturbation procedure and a local search however, instead of updating a single solution, it maintains a set of candidate solutions, each member of that set is compared to another one by a dissimilarity measure keeping the one with minimum value function. Computational results for the case of identical circles with up to $n = 100$ circles are given, whilst for the case of non-identical circles they considered 9 instances where the largest instance consists of 162 non-identical circles.

Liu et al. [51] use an approach based on energy landscape paving [27] which is an algorithm using low temperature Monte Carlo simulation that avoids being trapped in local minima via a penalty energy function for configurations that have already been explored. The improved energy landscape paving presented by Liu et al. [51] is an iterative approach whereby the centre of a chosen circle is repositioned at each iteration. They give computational results for their approach for the problem of packing up to 100 identical circles of unit radius into a containing circle, as well as for the problem of packing up to 162 non-identical circles into a containing circle. Liu et al. [52] extend Liu et al. [51] by introducing gradient descent into the approach. They give computational results for their approach for the problem of packing up to 50 non-identical circles into

a containing circle, as well as for the problem of packing up to 50 identical spheres of unit radius into a containing sphere. Liu et al. [50] make a number of modifications to the algorithm of Liu et al. [52]. They give computational results for their approach for the problem of packing up to 17 non-identical circles into a containing circle, as well as for the problem of packing up to 91 identical circles into a containing circle.

Huang et al. [40] propose an algorithm that is based on a quasi-physical approach. The quasi-physical approach considers that the container is of fixed size smaller than the best known solution, the items to be packed are elastic unit circles which are randomly allocated. The algorithm consists of three main procedures: a quasi-physical descent procedure, a quasi-physical basin-hopping procedure and an adjustment procedure. For the quasi-physical descent procedure the elastic circular items are pressed and pushed against each other until all disks stop moving, obtaining a local optima configuration. The idea of the basin-hopping procedure is to avoid being trapped in the local minima, hence during this process each disk is subject to three kind of forces: the elastic energy inherent in each circle, where the elastic force is interpreted as the sum of the squares of the overlapping depth between the incumbent circle and any other object, the attractive force exerted from the centre of the container to the centre of each circle and the non-contact repulsive force exerted by the pushed out circles. The final process of the algorithm is to adjust the radius of the container in order to obtain a feasible packing with the smallest radius possible for the container. They give computational results for $n = 1$ to $n = 150$ having found 37 out of 150 better packings than those in [72].

Very recently Stoyan and Yaskov [75] presented an algorithm that solves the packing problem of identical circles of fixed size inside a circular container considering forbidden fixed circular areas. Because of the nature of the feasible region a special way to generate initial solutions is provided. For the optimisation of the problem they use a modification of the Zoutendijk method, the original method generates improved feasible directions and optimizes on that direction, additionally they implemented a strategy that involves

active inequalities. Regarding their computational results we can see that every instance they have considered has been improved with their approach. The smaller case consider 158 circles whilst the largest instance contains 957 circles.

2.3.2 Rectangular container

Instances involving the packing problem with a rectangular container having length L and width W may have different approaches in relation with the objective function. In general these instances consider circles of fixed size (identical or non-identical) and a variable size container. As for the objective some of the examples found in the literature are:

- for the square container ($L = W$) minimise the length of the side, so minimise L
- for the rectangle container minimise the perimeter or the area, that is minimise $L + W$ or minimise LW respectively
- regard one dimension of the rectangle as fixed and minimise the other dimension, so for a fixed L minimise W, or for a fixed W minimise L. Problems of this type are often referred to as strip packing problems (SPP) or as the circular open dimension problem (CODP)

Regarding the rectangular container the most studied case is for the square container ($L = W$) and Szabo et al. [67] present a review of the work that has been developed with special consideration to the algorithms that represent a computer aided approach to justify the optimality of the packings. The methods investigated are: energy function minimization: for this approach the objective function represent an energy function and the points on the feasible region represent electrical charges which are repulsing each other. The billiard simulation algorithm follows the idea of the billiard game, it sets a random initial solution of points imposing a circle around it without overlapping, for this approach it is considered that circles have a moving direction and a speed,

the iterative process start the balls and at every iteration the swing of each ball is reduce until the packing becomes rigid. The perturbation method randomly allocates n points as initial solution, during the iterative process one circle at a time is perturbed inside a small square around the centre of the circle whiting a distance 0.25 and the location of the nearest neighbour is updated if the distance between them has increased, this process is repeated until there are no more circles to move. The algorithm called "Threshold Accepting Modified Single Agent Stochastic Search for Packing Equal Circles in a Square" (TAMSASS-PECS) is based on the Threshold Accepting approach which is an algorithm very similar to simulating annealing. The algorithm starts with a random initial solution, a standard deviation and a threshold level. The algorithm improves the initial solution using local search and it uses as a stopping rule the value of the standard deviation. A deterministic approach based on LP relaxation is mentioned and more detail of a modified billiard simulation approach is given. They present as well an algebraic representation for an optimal packing for $n = 2$ to $n = 100$. Finally they consider the interval analysis method through which they validate the optimal solutions found. They present optimal packings for $n = 28, 29$ and 30 circles. Later on the work is extended in [76] by giving a more detailed description of the methods studied in [67] additionally providing the program codes. They give computational results for $n = 2$ to $n = 200$ that at that time (2007) were the best-known.

Huang and Ye [39] propose an algorithm that is based on a quasi-physical strategy presented in [41], this strategy aims to find a criterion under which n rigid objects are arranged inside a container with a predetermined shape, in order to do that they consider the container and the objects to be packed as elastic, they find a solution by using the laws of motion applied between the items and the container until the items and the container have their original shape. The Huang et al. [39] algorithm consists of three main processes: a local search, the so called "greedy vacancy search" and an early stop strategy. The local search is an iterative process that stops when a feasible

solution is found. They transform the original problem to an unconstrained problem where the objective function depends on the area of the square and on a penalty function (the "elastic energy" function defined from the non-overlapping constraints between the circles and the container). The greedy vacancy search is used as a perturbation method to exploit the biggest space left in a current solution. The idea is to extract one circle from its current position to the biggest free space considering this as the current initial solution for the local search. Their early stop strategy evaluates if the feasible solution obtained is a candidate that can be improved, if it cannot be improved then it is disregarded. The algorithm moves from the greedy vacancy search to the local search procedure only when a feasible solution is found, hence obtaining a better solution, or at least not worse. They give computational results from $n = 1$ to $n = 200$ presenting 17 out of 200 improvements over the best known packing at that time.

Machado and Leitao [57] propose an algorithm that solves the packing problem of identical circles inside a square container, their approach is based on genetic algorithms, from this point of view an individual is represented with two chromosomes; one is a vector with the coordinates of a solution for the packing problem whereas the second chromosome is a mating fitness function to evaluate the potential mates of the individual. This approach relies in the observation that in some cases the optimal configuration of an instance with n circles is also an optimal configuration with $n-i$ with $i \in [1, n-2]$ circles, this information is used to assess the performance of an individual for mating selection. They give computational results for $n = 2$ to 24 identical circles comparing their best solutions from four evolution mating selection functions and the optimal solutions. The accuracy considered for the results was four decimal places. No computation time is given for any approach.

2.3.3 Other containers

Even though the circular and square containers are the most studied instances for the packing of identical circles, Melissen [58, 59] provides optimal (proven) packings for up to 12 identical circles inside an equilateral triangle container. Afterwards Melissen and Schuur [60] use an approach based on simulated annealing, quasi-Newton method and human intelligence strategy for the solution of 16, 17 or 18 identical circles inside an equilateral triangle. Another contribution considering an equilateral triangle container is by Graham et al. [19] giving a dense packing for $n = 22$ to 34 identical circles. Other containers such as a right-angle-isosceles triangle and a semicircular container are found in [54].

2.4 Packing non-identical circles

This section considers the case of packing non-identical circles within circular and rectangular containers. In Section 2.4.1 we focus on work that considers a circular container whilst in Section 2.4.2 those concerning a rectangular container.

2.4.1 Circular container

Some early approaches to solve the packing problem of non-identical circles based on a quasi-physical method are presented in [38, 42]. Wang et al. [79], incorporate a quasi-physical and a quasi-human approach; the algorithm presented alternates from the quasi-physical to the quasi-human strategy until a solution is found or the time allowed has been reached. The physical strategy is based on the laws of motion and is used to obtain a packing of n circles until a local minima has found. The human strategy is based on the behaviour of people in a non satisfactory position, the most unfortunate will tend to move out as soon as possible while the more fortunate may wish to spread their wealth to obtain some balance. Under this considerations the human strategy aims to improve the

current solution by picking up the circle with the maximum pain (depth of overlap, in the human context the most unfortunate) and randomly allocate it inside the container. An improved version of the later is presented by Huang et al. [36] by changing the condition that allows switching from the quasi-physical to the quasi-human strategy in a shorter period of time; instead of considering a local minima solution as the condition to switch they change it by a promising local minimum solution. They give numerical results for four instances where the time was reduced by significant factors.

Zhang and Huang [81] propose a heuristic based on simulated annealing combined with heuristic strategies to avoid being trapped in a local minima. During the simulated annealing process the temperature may drop to zero meaning that the algorithm may get stuck in a local optima hence, in this stage of the algorithm heuristic strategies are used. The heuristic strategy implemented consists of grouping the circles according to their size, if overlaps are found within a group of circles then one of the embedded circles is picked out and randomly placed around the circular container; but if there are no overlaps by group, then the circle with the most embedded measure is randomly reallocated inside the container. They give computational results for four instances of which three are of different sized circles and one of equally sized circles.

Later on Zhang and Deng [80] proposed an algorithm that combines two well known heuristics: simulated annealing and tabu search. Let us recall that simulated annealing is a heuristic technique developed to approach global optimal solutions, in this algorithm simulated annealing is used during the optimisation process to escape from a local minima by allowing worse solutions to be accepted according to a probability function. On the other hand the idea of the tabu search technique is to forbid movements to prevent cycles and to guide the search to explore new neighbourhoods. They give computational results for nine instances, five of them consist of identical circles while the other four consist of non-identical circles.

In 2006 a contest was held for finding the optimal solutions for the packing problem

for unequal circles with radii $R_i = i$ for all $i = 1, ..., n$ inside a circular container of minimum radius. The instances considered were from $n = 5$ to $n = 50$. In Addis et al. [1] they present the algorithm that made them the winners of that contest. Their approach is based on population basin hopping [24] and the reduction of the space of variables. Although they won the contest further computational results are frequently reported. Müller et al. [64] addressed same problem using the simulated annealing approach reaching the best known solutions for $n = 5$ to 23 and 25 circles while beating the (previously) best known for $n = 24$ and for $n = 26$ to 50 circles. As mentioned previously above, the website [72] maintains the best solutions for different variants of the packing problem and is continually updated.

Huang et al. [37] present two heuristics to solve the packing problem of non-identical circles aiming to minimise the radius of the circular container. The first heuristic is based on the maximum hole degree strategy. The hole degree strategy is a measure of the benefit of placing a circle in the container in relation with those already packed hence, a maximum hole degree for any given circle represent the maximum benefit for placing that given circle inside the container. The second heuristic uses a self look-ahead strategy to improve over the solution obtained by the first heuristic. The self look-ahead strategy consists of examining every corner placement, that is, it places a circle tangent to at least two circles without overlapping and it measures the benefit of that corner placement by examining the relation between all circles inside and outside the container. Akeb et al. [4] extended the work in [37] by combining two strategies: the maximum hole degree and the minimum damage. Both strategies are used to determine the position of the next circle to be packed. Depending on the current solution the circle will be located according to one or the other strategy. Akeb et al. [2] propose an algorithm that embeds beam search within the dichotomous local search of the minimum radius of the container. The beam search branches out a node based on the maximum hole degree measure.

Lu et al. [56] propose an algorithm to solve the circle packing problem combining two strategies: Prune-Enriched-Rusenbluth Method (PERM) with the maximum cave degree approach [37]. PERM was conceived under a chemical point of view to generate a chain of polymers with large length [21]. Briefly, it is a method where the configurations formed are assigned a weight, if the weight of a partial configuration is less than a lower threshold it is pruned and it is enriched (enlarged) if the weight is greater than a given upper threshold. For the implementation given by Lu et al. the circles are packed one by one as close as possible according to the maximal cave degree approach forming partial configurations. PERM strategy is used in every partial packing to decide which branches to prune or to enrich based on the associated estimated weight that every partial configuration has. They present results for 14 instances where the radii of the circles to be packed are equal and unequal, the smallest instance consists of 7 equal circles whilst the largest instance consists of 90 equal circles.

Hifi et al. [32] present a heuristic that dynamically updates the centre and the radius of the circular container. The updating process comes after finding the best position for the current circle. Afterwards, Hifi et al. [31] propose an algorithm that combines adaptive and restarting techniques. The algorithm consist of three phases; a dynamic search that determines a packing solution, an adaptive phase that uses intensification and diversification on the solution obtained in the dynamic search (the intensification aims to find a smaller radius for the container whilst the diversification uses different packing techniques for the current solution) and finally the restarting phase that is based on the hill climbing approach.

Al-Mudahka et al. [5] propose an algorithm that combines two procedures: local search and a nested partition of the feasible space. The local search procedure used is tabu search, the tabu list is used to find a permutation of the circles that leads close to the optimal one, then they explore the feasible set by making a partition of it. As part of their nested partition procedure they use a polar reformulation of the problem as a

way to escape from a local optimum. The computational results given for their approach for the problem of packing non-identical circles inside a circular container consider two different instances; when the radii of circles is given by $R_i = i \ \forall i = 1, ..., n$ for up to $n = 50$ circles and when the radii is given by $R_i = 1/\sqrt{i} \ \forall i = 1, ..., n$ for up to $n = 35$ circles. As well they consider the case of identical unitary circles and give computational results for that case for up to 100 circles.

In order to solve the layout design of satellite module Liu et al. [48, 49] consider the two dimensional packing problem adding equilibrium constraints. In [48] the heuristic presented is based on simulated annealing while in [49] the heuristic presented is based on tabu search.

2.4.2 Rectangular container

As mentioned in Section 2.3.2 we have found (in published work) different ways in which the packing problem with a rectangular container has been studied, in this section we present a few of them considering non-identical circles.

Some work presented in the field of the packing problem has been directly motivated by industrial applications. George et al. [17] consider the problem of "fitting pipes of different diameters into a shipping container", the authors decided to tackle the problem through the two-dimensional packing problem considering that the circles and the rectangular container have fixed size. They formulate the packing problem as a mixed-integer non-linear programming problem: continuous variables represent the (x, y) coordinate position while binary variables take the value one if the circle is placed in the rectangle and zero otherwise. The objective function aims to maximise the total density provided by the circles packed. The binary non-linear constraint model is formed by considering that boundaries of the two types of constraints (those that guarantee circles are inside the container and the non-overlapping constraints) are multiplied by the binary variables. They propose an adaptive heuristic based on strategies found in genetic algorithms that

uses a stable solution. A stable solution is defined as one where the circles satisfy conditions such as: touching the bottom of the container, or one of the sides of the container or when the circles are resting on top of another circle as big as itself or on top of two resting circles. For the series of implemented heuristics it is important to obtain a stable solution considering that ultimately the circles packed in the rectangular container represent pipes being fitted in a shipping container. The heuristic algorithms produced stable or unstable solutions depending on the rules for locating the circles. The six heuristics presented were tested on a series of 66 test problems. The problems were generated by considering that any instance is formed with elements of one, two or the three different sets in which the circles were categorized: small, medium and large. The comparisons suggested that overall the best performance was produced by a quasi-random procedure presented when considering quality of solution and computational time.

Stoyan and Yaskov [74] address the circular open dimension problem (CODP) by considering a strip container of fixed width aiming to minimise the length of the container using the reduced gradient method which is another method to generate improving feasible directions, and the active inequality collection strategy that determines a working set of constraints that will be treated as equality constraints. Their approach consists of two phases: in phase one the radii of the circles are considered fixed and the aim of this phase is to find a local minima, in phase two the radii of a selected pair of circles are considered as variables, an increase of size for one circle and decrease for the other will lead to an improvement in the solution, jumping from one local minima to another until all the suitable pairs of circles are finished. They assessed their algorithm using a set of 30 instances giving computational results for only a subset of those used, the size of the problems presented range from $n = 20$ to 35 circles. In the conclusions it is suggested to use a multi-start strategy as the final result is sensitive to the initial solution.

Akeb et al. [3] address the CODP as in [74] but they consider non-identical circles of fixed radii. They present two version of an algorithm that is based on the augmented

beam search method. The first version is based on multi-start strategy, that is to restart the algorithm from a new initial solution to diversify the exploration of the feasible region aiming to find a global optima, the second version uses multi-start and separated beam strategies. "A beam search is a heuristic search technique that combines elements of breadth-first and best-first searches. Like a breadth-first search, the beam search maintains a list of nodes that represent a frontier in the search space. Whereas the breadth-first adds all neighbours to the list, the beam search orders the neighbouring nodes according to some heuristic and only keeps the n best, where n is the beam size" [71]. They test their algorithm on two sets of instances: one set is composed of instances presented in the literature and the second set comprises of 1560 instances randomly generated in a certain fashion. According to their results their algorithm dominates those against which it was compared.

Hifi et al. [34] present an algorithm to solve the constrained and unconstrained circular cutting problem where a rectangular surface with length L and width W is to be cut in non-identical circular pieces. The approach to solve it is via simulated annealing aiming to find the maximum area covered by the circular items without overlaps. They compare their work against a modified version of the original algorithms presented in [6, 8, 25] by adapting them to pack circular items, when originally they were devised to pack items other than circles. Additionally they compare their results against those in [74] and to those extracted somehow in [20]. Their computational results are presented in terms of the percentage coverage of the area of the rectangular container, they claim to have produced good solutions when contrasting against those found in the literature.

Hifi et al. [30] consider the circular cutting problem for a rectangular container. The rectangular container of length L, width W and the size of the circular items are fixed. The authors consider a maximum demand on certain circular pieces of a specific size. They developed two algorithms: one is based on constructive heuristics while the other is based on genetic algorithms. The instances considered are from [73], the solutions

given from the two heuristics developed were compared against solutions in [34, 73] and dominated those in [34, 73].

2.5 Formulation space search

Formulation space search is a relatively recent method that emerged with the need to escape from solutions that are stationary points and reach better solutions when considering non-linear non-convex problems. Stationary points are those whose derivative is zero but are neither minimum nor maximum. In Mladenović et al. [62] they observed that different but equivalent formulations of the same problem may have different characteristics that can be exploited to escape from stationary points and possibly find better solutions. That is, while in one formulation we reach a stationary point, this may not be the case in another formulation, hence a natural manner to proceed is swapping between formulations when a stationary point has been reached.

Under this framework Mladenović et al. [62] proposed the formulation space search for the circle packing problem considering two formulations of the problem: one in a Cartesian coordinate system, one in a polar coordinate system. Their algorithm solves the problem with one formulation at a time and when the solution is the same for all formulations, then the algorithm terminates. They considered the case of packing identical circles and their computational results are for up to 100 circles packed into the unit circle and the unit square.

Afterwards Mladenović et al. [63] improved on [62] by considering a mixed formulation of the problem; they set a subset of the circles in the Cartesian system whilst the rest of the circles are in the polar system. A reduction in the number of the non-overlapping constraints is made at the initial solution by disregarding points sufficiently far away from each other. They give computational results for up to 100 identical circles inside the unit circle.

Formulation space search is a new and relatively unexplored idea in the literature.

It has been applied to only a few problems in the literature additional to circle packing (e.g. timetabling, Kochetov et al. [43]). More discussion as to formulation space search can be found in Hansen et al. [26]. A related approach is variable space search, which has been applied to graph colouring (Hertz et al. [28, 29]). More recently formulation space search has been applied to the cutwidth minimization problem, Mladenovic et al. [66]. As well it has been used to present a strategy to solve mixed-integer non-linear programming problems in López and Beasley [53].

2.6 Conclusions

In this chapter we gave a description of the two-dimensional packing problem. We presented a brief historical note highlighting appropriate examples of the two-dimensional packing problem. We reviewed some of the most relevant work published in recent years for the packing problem of identical and non-identical circles, mainly considering circular and rectangular containers and briefly mentioned the work carried on with triangular containers. We also presented relevant literature related to formulation space search.

Chapter 3

Packing identical circles inside a circular container

In this chapter we address the packing problem of identical circles inside the unit circle container. In Section 3.1 we describe the dual nature of the problem and justify how it can be addressed by two different points of view. Section 3.2 presents the mathematical model used that represents the packing problem to be solved. In Section 3.3 we describe in detail the procedures implemented with formulation space search for the packing problem with identical circles. In particular, we explain the need to consider two procedures that tackle two key components of our heuristic: in one procedure we reduce of the size of the set of the non-overlapping constraints whilst with another procedure we ensure we obtain feasible solutions. In Section 3.4 we present the pseudocode, in Section 3.5 we start giving a glance of the algorithm with the use of pictures for a complete iteration. We present the computational results produced by our heuristic and assess its performance by comparing it with the best-known solutions [72], with approaches based on FSS [62, 63] and using some modifications to the formulation of the problem, we also compare with other work found in the literature [9, 23, 51, 68]. In order to investigate more about the behaviour of our FSS heuristic we consider in Section 3.6 some other alternative strategies to improve

(if possible) our results. Finally we finish this chapter with the conclusions in Section 3.8.

3.1 The circular container

Let us recall from Section 2.1 that the packing problem is concerned with the arrangement of n circles inside a bounded region. In this chapter the bounded region is the circular container which is the most studied case in the packing problem area. In the literature it is often addressed from two different (but equivalent) points of view:

a) to maximise the radius associated with the n circles when the container size (area) is fixed

b) to minimise the size (area) of the container to accommodate n circles of fixed radius

Justification: These two viewpoints are in fact equivalent because there exists a one-to-one relationship (based on scaling) between them. Consider, for the purposes of illustrating this equivalence, the container being a circle centred on the origin.

Let us assume that first point of view is true, that means that we have an optimal solution with circles centred at (x_i, y_i) $i = 1, ..., n$ with n identical circles with radii of maximum value, say R within a container of fixed size, without loss of generality let us assume that the container radius is equal to 1. If we scale the current solution $((x_i, y_i)$ $i = 1, ..., n)$ by a factor, say $1/R$, the new scaled solution that represents the coordinate centres for the n circles to be packed is obtained by multiplying by the scaled factor $(x_i/R, y_i/R)$ with circles that have a fixed radius 1 within a circular container with radius $1/R$. For this problem it is required to obtain a minimum size (area) for the circular container. As the area of a circle keeps a direct square relation with respect to its radius then minimising the radius $(1/R)$ is equivalent to minimising the area of the container.

3.2 The mixed formulation model

The mathematical model given in equations (3.1)-(3.11) describes the mixed formulation for the two-dimensional packing problem of identical circles maximising the radii of n circles inside the unit circle container. The packing problem is solved by our implementation of formulation space search. The container is centred at the origin of the Cartesian plane and concerning the notation used we have:

- C the set of indices of circles whose centres are expressed in Cartesian coordinates, so for circle $i \in C$ its centre is at (x_i, y_i) in Cartesian coordinates
- P the set of indices of circles whose centres are expressed in polar coordinates, so for circle $i \in P$ its centre is at (r_i, θ_i) in polar coordinates (where $C \cap P = \emptyset$ and $C \cup P = \{1, ..., n\}$)
- Q the set of all pairs $\{(i, j) \mid i = 1, ..., n;\ j = 1, ..., n;\ i \neq j\}$, so $|Q| = n(n-1)/2$
- R the radius associated with each of the n circles
- $R_{overlap}$ an upper bound on R, formally $R_{overlap}$ is the maximum radius that the circles can have before they must overlap due to area considerations, defined here by equating the area of the containing unit circle ($\pi 1^2$) to the total area of the n circles ($n\pi R^2_{overlap}$), so $R_{overlap} = 1/\sqrt{n}$

Although we have above (using disjoint sets C and P) separated centres expressed in Cartesian and polar coordinates note here that the relationship between the two coordinate systems is that a point (x, y) in Cartesian space has equivalent coordinates (r, θ) in polar space where $x = r\cos(\theta)$ and $y = r\sin(\theta)$.

The formulation used is:

$$\max \quad R \tag{3.1}$$

st

$$x_i^2 + y_i^2 \leq (1 - R)^2 \qquad \forall i \in C \tag{3.2}$$

$$r_i \leq 1 - R \qquad \forall i \in P \tag{3.3}$$

$$(x_i - x_j)^2 + (y_i - y_j)^2 \geq 4R^2 \qquad \forall (i,j) \in Q \text{ with } i,j \in C\ i < j \tag{3.4}$$

$$(x_i - r_j \cos(\theta_j))^2 + (y_i - r_j \sin(\theta_j))^2 \geq 4R^2 \qquad \forall (i,j) \in Q \text{ with } i \in C\ j \in P \tag{3.5}$$

$$r_i^2 + r_j^2 - 2r_i r_j \cos(\theta_i - \theta_j) \geq 4R^2 \qquad \forall (i,j) \in Q \text{ with } i,j \in P\ i < j \tag{3.6}$$

$$X_i - \Delta \leq x_i \leq X_i + \Delta \qquad \forall i \in C \tag{3.7}$$

$$Y_i - \Delta \leq y_i \leq Y_i + \Delta \qquad \forall i \in C \tag{3.8}$$

$$0 \leq r_i \leq 1 \qquad \forall i \in P \tag{3.9}$$

$$0 \leq \theta_i \leq 2\pi \qquad \forall i \in P \tag{3.10}$$

$$0 \leq R \leq R_{overlap} \tag{3.11}$$

The objective, equation (3.1), maximises the radius associated with the circles. Equations (3.2) and (3.3) are the constraints which ensure that every circle is fully inside the container, in this case the unit circle. Notice here that whilst equation (3.2) is a non-linear equation when expressed in Cartesian form, it is a linear equation, equation (3.3), when expressed in polar form.

Equations (3.4)-(3.6) ensure that no circles overlap each other. Equation (3.4), for example, says that the Euclidean distance between the centres of circle i and circle j, $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ must be at least $2R$ (since each circle is of radius R). Note here that for computational reasons we (as is common in the literature) have squared both sides of this constraint to eliminate the square root. Equations (3.5)-(3.6) are as equation (3.4), but where one or both of the circles has its centre expressed in polar coordinates.

Equations (3.7)-(3.11) are the limits on the variables. Equations (3.7) and (3.8) limit variables for circles expressed in the Cartesian system, where X_i and Y_i are known values taken from the initial solution with Δ a range of movement. ***Note here that although equation (3.9) can be deduced from equation (3.3), it is given here for completeness***. Equation (3.10) limits the polar angle, so avoiding a multiplicity of equivalent solution values, $\theta_i + k(2\pi)$ for all integer values of k.

The difference between the formulation used for our heuristic algorithm and those given in [62, 63] are:

- an estimated upper bound denoted as $R_{overlap}$ for variable R
- we introduced the Q set for the non-overlapping constraints
- we use limits for circle centres that are expressed in the Cartesian system as given in equations (3.7) and (3.8)

3.3 The formulation space search heuristic

The heuristic developed for packing identical circles is based on formulation space search and we will denote it as "FSS". It considers two processes that influence the quality of the final solution. The first process is related to how we decrease the size of the set of the non-overlapping constraints ($|Q|$) whilst the second process amends any non-accurate solution (if any) disregarded by our non-linear solver (Snopt [18, 35]).

Regarding the size of the set of the non-overlapping constraints (equations (3.4)-(3.6)), it is formed by pairs of circles and assuming that we have n circles there are $\binom{n}{2} = n(n-1)/2$ possible ways to create them, that is $|Q| = n(n-1)/2$. In other words, the more circles the heuristic is dealing with the larger the cardinality of Q, leading to an exhausted and possibly unfruitful search for feasibility considering pairs of circles that are not even close, e.g., those whose distance between them is much greater than $2R$. The idea to decrease the size of Q has already been explored and implemented in

different ways in [10, 62]. In Section 3.3.1 we propose a new procedure to determine the pair of circles that will be elements of Q.

Numerical accuracy is particularly important for problems of this kind, the best-known results given at the Packomania web site of Specht (available at [72]) are given to a very high degree of accuracy (30 decimal places). In order to address numerical accuracy we have implemented in our heuristic a process (called "correction step") that ensures feasibility for any given solution by the non-linear solver when expressed with a high degree of precision. Details of this process are given in Section 3.3.3.

3.3.1 The set of the non-overlapping constraints

As mentioned earlier, in order to reduce the number of non-overlapping constraints we need to reduce the size of Q. This is done by introducing linear constraints on the range of possible values for centre coordinates, that is, for each circle i having a known point (X_i, Y_i), the centre of the circle is not allowed to move more than Δ from this point at the next iteration of our FSS heuristic. For ease of presentation all centres are expressed here in Cartesian coordinates.

Given these restrictions the constraints on the centre coordinates of circle i become

$$X_i - \Delta \leq x_i \leq X_i + \Delta \text{ and } Y_i - \Delta \leq y_i \leq Y_i + \Delta \tag{3.12}$$

i.e. $x_i \in [X_i - \Delta, X_i + \Delta]$ and $y_i \in [Y_i - \Delta, Y_i + \Delta]$. Consider the non-overlapping constraint equation (3.4). This involves the term $(x_i - x_j)^2$ for two circles i and j, whose centres now have a restricted range, so we can deduce the minimum value that this term can take. Given the ranges, $x_i \in [X_i - \Delta, X_i + \Delta]$ and $x_j \in [X_j - \Delta, X_j + \Delta]$ these two ranges overlap (in other words x_i can equal x_j) if and only if one of the end points lies in the other range. In other words these ranges overlap if:

$$\begin{aligned} X_i - \Delta \in [X_j - \Delta, X_j + \Delta] \text{ or } X_i + \Delta \in [X_j - \Delta, X_j + \Delta] \text{ or} \\ X_j - \Delta \in [X_i - \Delta, X_i + \Delta] \text{ or } X_j + \Delta \in [X_i - \Delta, X_i + \Delta] \end{aligned} \tag{3.13}$$

If the ranges overlap then the minimum value of $(x_i - x_j)^2$ is zero. If the ranges do not overlap then the minimum value of $(x_i - x_j)^2$ will occur when x_i and x_j take values at the end of their ranges, so in this case the minimum value of $(x_i - x_j)^2$ will be:

$$\begin{aligned}\min\{&[(X_i - \Delta) - (X_j - \Delta)]^2,\ [(X_i - \Delta) - (X_j + \Delta)]^2,\\ &[(X_i + \Delta) - (X_j - \Delta)]^2,\ [(X_i + \Delta) - (X_j + \Delta)]^2\}\end{aligned} \tag{3.14}$$

Working out every element of expression (3.14), we have that:

$$\begin{aligned}[(X_i - \Delta) - (X_j - \Delta)]^2 &= [(X_i + \Delta) - (X_j + \Delta)]^2 = (X_i - X_j)^2\\ [(X_i - \Delta) - (X_j + \Delta)]^2 &= (X_i - X_j)^2 + 4\Delta(X_j - X_i + \Delta)\\ [(X_i + \Delta) - (X_j + \Delta)]^2 &= (X_i - X_j)^2 + 4\Delta(X_i - X_j + \Delta)\end{aligned} \tag{3.15}$$

Hence, if we denote $d_x(i,j) = (X_i - X_j)^2$ then expression (3.14) is equivalent to:

$$\min\{d_x(i,j), d_x(i,j) + 4\Delta(X_j - X_i + \Delta), d_x(i,j) + 4\Delta(X_i - X_j + \Delta)\} \tag{3.16}$$

A pseudocode to computing the minimum distance between any two circles i and j considering only the x-coordinate is given in algorithm 3.1. Let us note that the procedure to compute the distance over the y-coordinate is exactly the same.

In a similar manner we can compute the minimum value of the $(y_i - y_j)^2$ term in equation (3.17) as zero if:

$$\begin{aligned}&Y_i - \Delta \in [Y_j - \Delta, Y_j + \Delta] \text{ or } Y_i + \Delta \in [Y_j - \Delta, Y_j + \Delta] \text{ or}\\ &Y_j - \Delta \in [Y_i - \Delta, Y_i + \Delta] \text{ or } Y_j + \Delta \in [Y_i - \Delta, Y_i + \Delta]\end{aligned} \tag{3.17}$$

and as:

$$\begin{aligned}\min\{&[(Y_i - \Delta) - (Y_j - \Delta)]^2,[(Y_i - \Delta) - (Y_j + \Delta)]^2,\\ &[(Y_i + \Delta) - (Y_j - \Delta)]^2,[(Y_i + \Delta) - (Y_j + \Delta)]^2\}\end{aligned} \tag{3.18}$$

Algorithm 3.1 Computing minimum distance pseudocode

Function $d_{\min_x}(i,j) \leftarrow \min_x(X, Y, \Delta)$

if $X_i = X_j$ **then**

$d_{\min_x}(i,j) \leftarrow 0$

else if $X_i > X_j$ **then**

$d_{\min_x}(i,j) \leftarrow (X_i - X_j)^2 + 4\Delta(X_j - X_i + \Delta)$

else if $X_i < X_j$ **then**

$d_{\min_x}(i,j) \leftarrow (X_i - X_j)^2 + 4\Delta(X_i - X_j + \Delta)$

end if

otherwise. In a similar way, denoting $d_y(i,j) = (Y_i - Y_j)^2$ expression (3.18) can be simplified as:

$$\min\{d_y(i,j), d_y(i,j) + 4\Delta(Y_j - Y_i + \Delta), d_y(i,j) + 4\Delta(Y_i - Y_j + \Delta)\} \tag{3.19}$$

Let us consider Figure 3.1 to clarify what has been described above. In this figure we have two known points, (X_i, Y_i) and (X_j, Y_j), and the square surrounding each point indicates the range within which the centre's, (x_i, y_i) and (x_j, y_j), for circles i and j can be located. Deciding if the centres of the circles i and j may overlap we need to know how close the x and y coordinates of circles i and j may be by calculating the minimum values of terms $(x_i - x_j)^2$ and $(y_i - y_j)^2$. According to Figure 3.1 the minimum value of $(x_i - x_j)^2$ is determined by squaring of the length of the horizontal line that goes from point A to point B that are joining the left-hand square to the right-hand square, namely $[(X_i - \Delta) - (X_j + \Delta)]^2$. The minimum value of $(y_i - y_j)^2$ is zero (since the ranges overlap).

Now if the sum of these minimum values for $(x_i - x_j)^2$ and $(y_i - y_j)^2$ is $\geq 4R^2$ (compare equation (3.4)) then there is no need to impose a non-overlapping constraint for circles i and j, because they cannot overlap each other by definition given the ranges imposed.

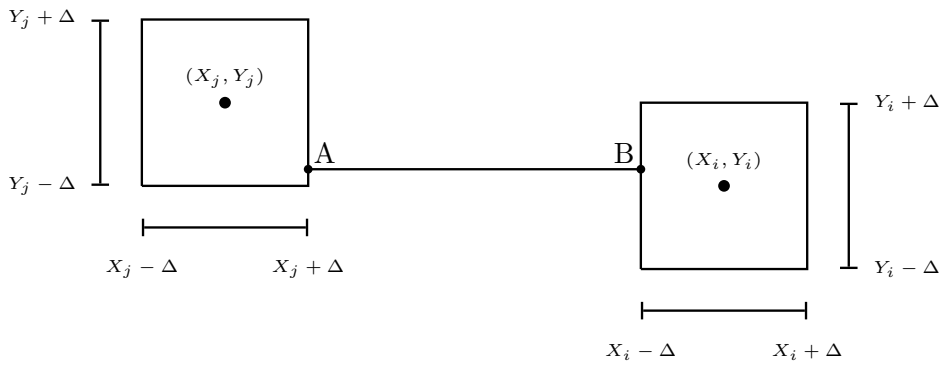


Figure 3.1: Example of how the non-overlapping constraints are determine

With reference to Figure 3.1 then given the square ranges allowed for circles i and j the best possible position for these circles (each of radius R) in any attempt to have them overlap would be centre them at points A and B respectively. If the distance from A to B is $\geq 2R$ (equivalently the squared distance is $\geq 4R^2$) then there is no need to impose a non-overlapping constraint for circles i and j.

Clearly we do not know R, since that is a variable in the optimisation, but we can replace it by any valid upper bound which we do know, here we use $R_{overlap}$. Hence if the sum of these minimum values is $\geq 4R^2_{overlap}$ circles i and j cannot overlap and the pair (i, j) need not be included in Q. Conversely if this sum is $< 4R^2_{overlap}$ the pair (i, j) does need to be included in Q.

Hence, in summary here, we take all pairs (i, j) of circles, do the calculation (equations (3.13)-(3.18)) outlined above (which is computationally an easy task) and thereby identify the pairs of circles that need to be included in Q.

We denote the above procedure as $OverlapSet(X, Y, \Delta, R_{overlap}, n)$ that returns the set Q of pairs of circles, it is also described in pseudocode 3.2.

We would extend here our previous comment that other authors in the literature have also attempted to reduce the number of overlap constraints. Mladenovic et al [63] reduce the number of constraints by ignoring overlap constraints for circles whose cen-

Algorithm 3.2 Reducing the size of Q pseudocode

Function $Q \leftarrow OverlapSet(X, Y, \Delta, R_{overlap}, n)$

$Q \leftarrow \emptyset$

for $i = 1$ to $n - 1$ **do**

  for $j = i + 1$ to n **do**

    $d_{\min_x}(i, j) \leftarrow min_x(X, Y, \Delta)$ {call function $\min_x$}

    $d_{\min_y}(i, j) \leftarrow min_y(X, Y, \Delta)$ {call function $\min_y$}

    if $d_{min_x}(i, j) + d_{min_y}(i, j) < 4R^2_{overlap}$ **then**

      $Q \leftarrow Q \cup (i, j)$ {update Q set}

    end if

  end for

end for

tres (at the initial solution, before optimisation) are sufficiently far apart. In our view the disadvantage of their approach is that during the optimisation process they do not consider a constraint (3.12) that limits the location of a new solution for circles in the Cartesian system in order to take advantage of the previous solution to obtain a better solution (unlike our approach above). An important difference therefore between our approach and earlier formulation space search work for circle packing (such as Mladenovic et al [62] who did not reduce overlap constraints, and Mladenovic et al [63]) is the use of constraints that limit the range of movement for circle centres.

Birgin and Gentil [10] use a distinctly different approach to reduce the number of overlap constraints. They first replace the $n(n-1)/2$ overlap constraints, equations (3.4)-(3.6), by a single constraint which is the sum of $n(n-1)/2$ non-linear terms. Then, in order to reduce the computational effort relating to evaluating this constraint (which they may need to do many times during the course of a non-linear solution algorithm), they consider a partition of the container into regions in such a way that circles whose

centres are not in the same (or an adjacent) region cannot contribute to the constraint. Essentially therefore their approach consists of replacing $n(n-1)/2$ overlap constraints, each of which is computationally inexpensive to evaluate numerically (requiring O(1) operations), by a single surrogate constraint that computationally requires O$(n(n-1)/2)$ operations to evaluate.

3.3.2 Optimisation problem

In our FSS heuristic the sets C and P are of approximately equal size. The initial solution is randomly generated at every iteration, however it is set in a slightly different way for initialization step and for the remaining iterations of our heuristic algorithm. In iteration one we randomly generate the initial solution using polar coordinates ($r_i \in [0,1]$ and $\theta_i \in [0, 2\pi]$), converted to Cartesian coordinates for those circles that belong to C and denoted as (x^0, y^0), note here that only for this iteration the known points $(X, Y) = (x^0, y^0)$, this procedure is also explained in algorithm 3.3. After we have obtained a solution from the non-linear solver, the initial solution for next iteration will be randomly generated at a Δ distance from the previous solution obtained by the non-linear solver. To clarify this point, let us consider (x, y) as the solver solution, which represent the centre coordinates of each circle, we set the known points $(X, Y) = (x, y)$ hence a new initial solution will be given by $x^0 \in (X - \Delta, X + \Delta)$ and $y^0 \in (Y - \Delta, Y + \Delta)$.

Limited computational experience indicated that, whilst we impose range constraints on all circles in terms of computing Q (as described above), it was sufficient purely to impose range constraints on circles whose centres are expressed in Cartesian coordinates in terms of the optimisation. This allows additional flexibility for positioning of circles whose centres are expressed in polar coordinates. Although this might result in some circles overlapping in the solution any such overlaps are corrected as explained in Section 3.3.3 below.

Hence (at each iteration) the non-linear optimisation problem that we solve is op-

Algorithm 3.3 First Initial Solution pseudocode

Function $(x^0, y^0, X, Y) \leftarrow InitialSolution(n, |C|)$

for $i = 1$ to n **do**

$r_i \leftarrow \text{random}[0, 1]$ {generate random values from}

$\theta_i \leftarrow \text{random}[0, 2\pi]$ {a uniform distribution for r and θ}

end for

$x_i^0 = r_i \cos(\theta_i)$ and $y_i^0 = r_i \sin(\theta_i)$ $\forall i \in C$ {initial solution for circle in C}

$x_i^0 = r_i$ and $y_i^0 = \theta_i$ $\forall i \in P$ {initial solution for circle in P}

$X = r\cos(\theta)$ and $Y = r\sin(\theta)$ {vector (X, Y) of known points}

timise (3.1) subject to (3.2)-(3.11). We denote this non-linear optimisation problem as $NLP(x^0, y^0, C, P, Q, X, Y, \Delta, R_{overlap})$ where x^0 and y^0 represent an initial solution for our optimisation solver Snopt in each iteration. In the computational results reported later below we set an initial solution for this solver by randomly generating $x_i^0 \in (X_i - \Delta, X_i + \Delta)$ and $y_i^0 \in (Y_i - \Delta, Y_i + \Delta)$ $(i = 1, ..., n)$.

3.3.3 Feasibility and correction step

As mentioned earlier, numerical accuracy is an issue that needs to be addressed to ensure that all solutions are feasible when expressed with high degree of decimal place accuracy. The correction process deals with feasibility, that is, it needs to verify two conditions and correct them when necessary: all circle centres are inside the container and that every pair of circles do not overlap. Let us consider (x_i, y_i) $i = 1, ..., n$ a given solution by the non-linear solver expressed in Cartesian coordinate system where R^* is the maximum radius associated with the optimiser solution.

First we must ensure that all circle centres are inside the container, in this case the unit circle, that is $x_i^2 + y_i^2 \leq 1$ $i = 1, ..., n$. If this condition is not satisfied for any particular circle we reposition it inside the container, this process is detail in algorithm 3.4.

Even though computational results have shown that circle centres outside the container is not a common situation we have to cover all possible situations to ensure feasibility for every solution.

Verifying the second condition involves the non-overlapping constraints. Let us set:

$$R = \min\left\{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}/2 \mid i = 1, ..., n;\ j = 1, ..., n;\ i < j\right\} \tag{3.20}$$

to ensure that the radius R is such so as to prevent circles from overlapping and then set:

$$R^* = \min\left\{R, \min\left\{1 - \sqrt{x_i^2 + y_i^2} \mid i = 1, ..., n\right\}\right\} \tag{3.21}$$

to ensure that all the circles are fully inside the unit circle. By the nature of the procedure adopted above, R^* must be associated with a feasible solution to the problem. Note here that the value of R^* may (because of the procedure above) differ (albeit possibly only slightly) from the radius value returned by the solver as the maximum possible radius.

We denote the above procedure as $Correction(x^0, y^0, x, y)$ and it returns a value for radius that must (by construction) be associated with a feasible solution. Pseudocode for this procedure is given in algorithm 3.4. In our computational implementation of this correction procedure we used a MATLAB function called vpa (which is the acronym for variable precision arithmetic) that gives as many digits of accuracy as we desire.

3.4 Pseudocode

In this section we present Algorithm 3.5, the pseudocode that represents the outline of our heuristic. In Section 3.4.1 we also present an example of the performance of our heuristics using 10 identical circles, showing pictures of the first iteration of the algorithm developed that will be described below.

Regarding the pseudocode, it is composed of two processes: Initialisation and Optimisation. In the initialisation process we set:

Algorithm 3.4 Correction step pseudocode

Function $(R^*, x, y) \leftarrow Correction(x^0, y^0, x, y)$

for $i = 1$ to n **do**

if $x_i^2 + y_i^2 > 1$ and $(x_i^0)^2 + (y_i^0)^2 \leq 1$ **then**

$(x_i, y_i) \leftarrow (x_i^0, y_i^0)$

else

$r_i \in [0, 0.99]$

$x_i = r_i \cos(\theta_i)$ and $y_i = r_i \sin(\theta_i)$

end if

end for

$R \leftarrow \min\left\{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}/2 \mid i = 1, ..., n;\ j = 1, ..., n;\ i < j\right\}$

$R^* \leftarrow \min\left\{R, \min\left\{1 - \sqrt{x_i^2 + y_i^2} \mid i = 1, ..., n\right\}\right\}$

- $|C| \leftarrow \lfloor n/2 \rfloor$ number of circles in the Cartesian system are approximately equal
- $R_{overlap} \leftarrow \frac{1}{\sqrt{n}}$, let us recall from Section 3.2 that $R_{overlap}$ is an upper bound for variable R based on area comparison depending of the number of circles (n) to be packed
- $\Delta \leftarrow \frac{2}{3} R_{overlap}$ factor involved in creating a range to allow centre coordinates to move at most Δ distance from a known point (X, Y), also it helps to create limits over variables expressed in Cartesian coordinate system during the optimisation process. Its value depends (see below) on the current best solution for R, so we use $R_{overlap}$ for the initial stage of the heuristic.
- $R_{best} \leftarrow 0$ initialising variable that will keep track of best value for R variable
- $t \leftarrow 0$ initialise the iteration counter
- $t\_rep \leftarrow 0$ initialise the replication counter

In Section 3.6.3 we explain and justify based on numerical experiments the choice for $\frac{2}{3}$ as the associated value to Δ, the number of replications for all instances as well as the number of iterations in each replication.

We generate a random initial solution by using polar coordinates ($r_i \in [0, 1]$ and $\theta_i \in [0, 2\pi]$), converted to Cartesian coordinates for those circles that belong to C and denoted as (X, Y).

Regarding the optimisation, it is an iterative process where we seek the maximum value R_{best} through different processes described in previous sections:

- *overlapSet* returns the pairs of circles (i, j) that form the elements of the non-overlapping constraints set Q
- NLP stands for the step where the solver returns a solution (x, y) of the best result found using solver Snopt
- *Correction* returns the corrected value (if necessary) for variable R as the best maximum radius common to all circles denoted as R^*

We also observed that a handful of results from the *Correction* step (in particular for cases with large number of circles) returned a small value for R^* such as 0.001, this indicates that there is at least one pair of centre circles whose distance apart is $2R^* = 0.002$. As the generation of the initial solution for next iteration depends on the Δ value, such a small value for R^* affects and leads to a smaller value for Δ, thus almost forcing the algorithm to remain with a solution that does not allow new points to be explored more than Δ from the current centres. Hence when this (although rare) situation happens in order to have a wider range of movement than, $\frac{2}{3}R^* = 0.0006$, we arbitrarily choose to use $\frac{1}{10}R_{overlap}$. We set a new iteration by assigning current non-linear solver solution (x, y) considering all their elements in their Cartesian form to a vector of known points (X, Y) that will be the centre of a square with distance Δ in order to generate a new initial solution $x^0 \in (X - \Delta, X + \Delta)$ and $y^0 \in (Y - \Delta, Y + \Delta)$

Algorithm 3.5 Formulation space search pseudocode

Function $(R_{best}, X_{best}, Y_{best}) \leftarrow FSS(n, replication\_limit, iteration\_limit)$

Initialisation: $|C| \leftarrow \lfloor n/2 \rfloor$ $\Delta \leftarrow \frac{2}{3}R_{overlap}$ $R_{best} \leftarrow 0$ $t\_rep \leftarrow 0$

repeat

$t \leftarrow 0$

$(x^0, y^0, X, Y) \leftarrow InitialSolution(n, |C|)$ {a first initial solution}

repeat

$Q \leftarrow OverlapSet(X, Y, \Delta, R_{overlap})$ {find the overlap set Q}

$(x, y, R) \leftarrow NLP(x^0, y^0, C, P, Q, X, Y, \Delta, R_{overlap})$

$(R^*, x, y) \leftarrow Correction(x^0, y^0, x, y)$ {correct the radius}

$R_{best} \leftarrow \max\{R_{best}, R^*\}$ {update R_{best}}

$(X_{best}, Y_{best}) \leftarrow (x, y)$ {save coordinates associated with R_{best}}

if $R^* \leq 0.001$ **then**

$\Delta \leftarrow \frac{1}{10}R_{overlap}$

else

$\Delta \leftarrow \frac{2}{3}R^*$ {update Δ}

end if

$t \leftarrow t + 1$ {update iteration counter}

$(X, Y) \leftarrow (x, y)$ {set (X, Y) to the current solution}

$x^0 \in (X - \Delta, X + \Delta)$ and $y^0 \in (Y - \Delta, Y + \Delta)$ {new initial solution (x^0, y^0)}

$C \leftarrow P$ $P \leftarrow \{1, ..., n\} \setminus C$ {switch the sets C and P}

until $t = iteration\_limit$

$t\_rep \leftarrow t\_rep + 1$ {update replication counter}

until $t\_rep = replication\_limit$

as explained in section 3.3.2. Once we have generated a new initial solution in terms of the Cartesian system we translate those that should be expressed in the Polar system.

3.4.1 A glance of our FSS

Figure 3.2 illustrates example steps from our FSS approach. Here there are four different pictures plotted using results from our heuristic when solving the packing problem for $n = 10$ identical circles inside the unit circle. Let us state that from now on, circles with a small "o" in their centre represent circles expressed in the Cartesian coordinate system while circles with an asterisk "$*$" in their centre are those expressed in the polar coordinate system. In Figure 3.2 we have not been included the solutions obtained by the correction step as they are the same as those given by the non-linear solver, this means that there were no overlaps.

Figure 3.2(a) represents a randomly generated initial solution inside the unit circle container with circles of radii zero as $R_{best} = 0$. The initial solution is the starting point for the solver to return a solution that in this case is graphically reproduced in Figure 3.2(b) with $R_{best} = 0.25936275$. As mentioned before, the solution given by the correction step turned out to be the same as that given by the solver $R^* = R_{best}$.

Next step in our heuristic can be seen in Figure 3.2(c), we switch those centre coordinates expressed in Cartesian to polar coordinate system and vice-versa, depicted in the same picture we have an initial solution for next iteration. Consider 3.12 where the known points (X, Y) are given by the last solution from the solver, $\Delta = \frac{2}{3}R^* = 0.1729333$, hence a range of movement (depicted as the green square around the known point) allows to have a close but new initial solution for next iteration, it is represented by the red small o and the red asterisk *. The returned solution by the solver is depicted in Figure 3.2(d) with $R^* = R_{best} = 0.26093107$ so an improved solution compared with Figure 3.2(b). Following this iterative manner we precede to obtain the best solution in iteration 48. The best solution is shown in Figure 3.2(e) with $R_{best} = 0.262258924129267$.

Figure 3.3 illustrates the variation in the value of R^*, these values were taken from the returned results from our heuristic composed of 80 iterations to pack 10 identical circles inside the unit circle. The graphic suggests that for this small problem the number

of iterations is more than required to obtain the best result but it gives more chances to explore points around the previous given solution by the solver to form the initial solution for next iteration.

3.5 Computational results

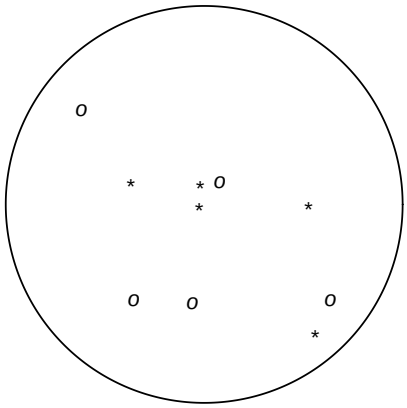
The results given in this section for our FSS heuristic were produced using an Intel Core 2 pc (2.26GHz, 4GB RAM). Our heuristic was coded in MATLAB 7.0 and as a subroutine used the non-linear optimisation solver Snopt [18, 35].

We consider different comparisons: In Section 3.5.1 we compare the results produced by our heuristic with those results presented in Packomania web site [72], Sections 3.5.2 and 3.5.3 present the comparisons made with previous work based on the FSS approach found in the literature. In Section 3.5.4 we compare with other work based on different approaches found in the literature and finally in Section 3.6 we include alternative strategies we considered aiming to improve over current solutions.

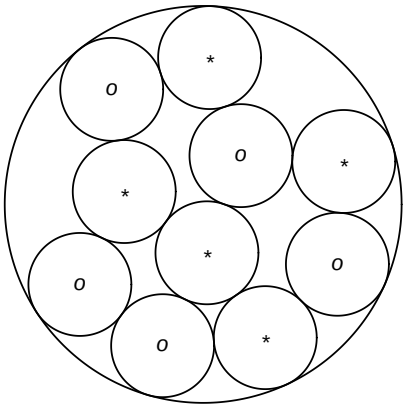
The measurement used to compare our results with others is called ***percentage deviation***, this measure describes how accurate are our results with respect to those compared. Let us denote f_{bk} as the best-known solution whilst f_b represents the best solution obtained by an approach, hence the ***percentage deviation*** is calculated as: $100(f_{bk} - f_b)/f_{bk}$.

We should be aware here that there are a number of complications in terms of comparing circle packing results presented in the literature:

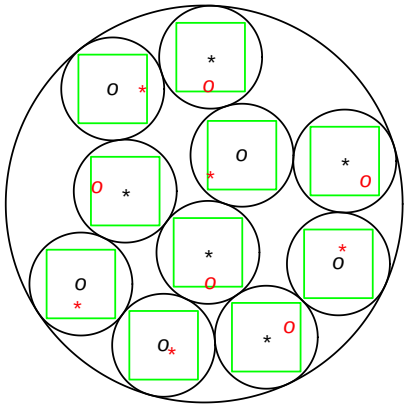
- some papers give results in terms of the circle radius, others in terms of the inverse of that radius
- the "Best-known" value is not a static value, rather it is dynamic as it may change over time as [72] is updated with improved results



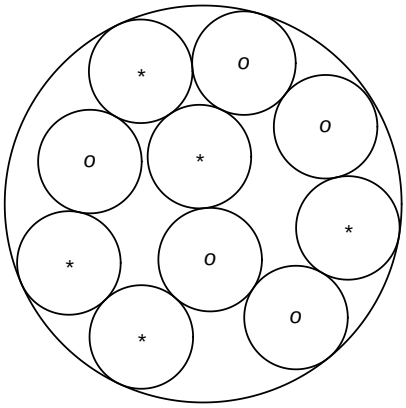
(a) Random initial solution



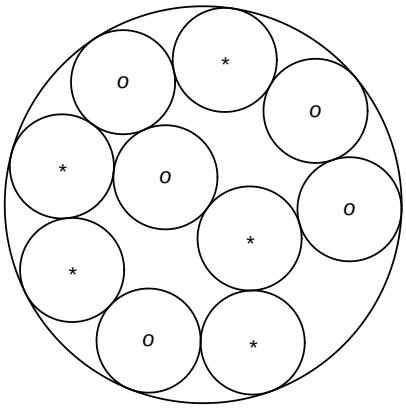
(b) Non-linear solver solution



(c) New swapped initial solution with centres * and o at a Δ distance from current solution with centres o and * respectively



(d) Non-linear solver solution



(e) Final solution

Figure 3.2: Snapshots for $n = 10$ circles

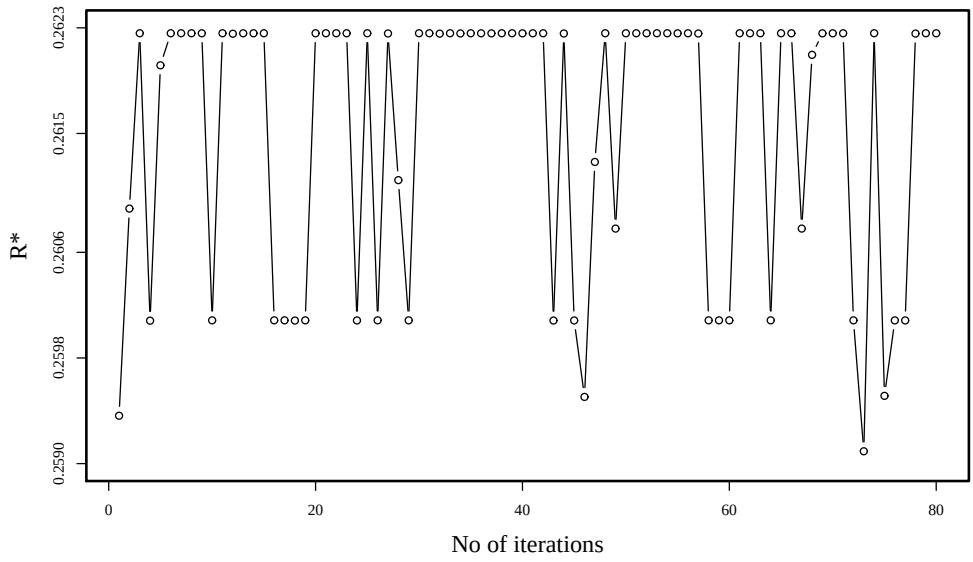


Figure 3.3: Variations of R^* for $n = 10$ identical circles

- different papers use different degrees of precision (number of decimal places given in the published results)
- some papers present results that are on closer inspection invalid (due to a lack of sufficient numeric precision, typically detected by comparing the detailed results given in the paper with [72])

3.5.1 Comparing with Packomania web site [72]

Let us recall that in Section 2.1 we stated that Packomania web site started reporting the best-known solution to the packing problem since 1999 and it is continually updating the best-known solutions, although they do not give computation time. Hence, we clarify that comparisons presented in Table 3.1 are based on the "best-known" solutions taken in January 2011.

Table 3.1 is composed of five columns: First column refers to the number n identical circles to be packed, second column refers to the best-known solutions taken from Packomania web site [72] in January 2011, third column shows the results produced

by our heuristic, fourth column displays the accuracy of our results with respect to the best-known by using the percentage deviation, finally fifth column gives the total time spent expressed in seconds. The results displayed on Table 3.1 represent a series of instances that take values for $n = 10$ to 100 circles in step of 5 followed by $n = 125, 150, 175, 200, 250$ and 500.

Table 3.1: Comparison with Packomania web site [72] for the unit circle container

n	Best-known solution	OUR Best solution	% deviation	Total time (sec)
10	0.262258924190	0.262258924188	0.000000001	40.59
15	0.221172539086	0.221172539085	0.000000001	36.97
20	0.195224011019	0.195224011010	0.000000004	47.30
25	0.173827661421	0.173827661389	0.000000019	64.77
30	0.161349109065	0.161349109023	0.000000026	81.61
35	0.149316776635	0.149316776574	0.000000041	112.36
40	0.140373604203	0.140373604193	0.000000007	109.38
45	0.132049594252	0.132049592409	0.000001396	150.98
50	0.125825489530	0.125825489530	0.000000001	173.06
55	0.121786324528	0.121786324528	0.000000000	218.58
60	0.115657480133	0.115657480132	0.000000001	245.97
65	0.110896743723	0.110896743719	0.000000003	288.75
70	0.107001616606	0.107001616367	0.000000223	347.45
75	0.103390915666	0.103390909387	0.000006073	407.75
80	0.100319499416	0.100319470202	0.000029121	478.99
85	0.098395063693	0.098395063689	0.000000003	621.22
90	0.094822059587	0.094822059542	0.000000048	586.38
95	0.092249177761	0.092249116345	0.000066576	692.78
100	0.090235200288	0.090235041265	0.000176232	809.86
125	0.080852343329	0.080852333325	0.000012373	1324.81
150	0.074289754450	0.074288436073	0.001774642	2225.91
175	0.068792158147	0.068782834112	0.013553922	3183.07
200	0.064669354186	0.064665524191	0.005922427	4814.65
250	0.057927485801	0.057926958793	0.000909771	9518.75
500	0.041437143525	0.041314938258	0.294917207	86117.3
Average			0.012694805	4507.97

The results produced by our approach were obtained by considering that the algo-

rithm is composed of 25 replications and each replication consist of 80 iterations having random initial solutions. Packomania web site keeps record of the best-known solutions to a high degree of accuracy (30 decimal places), for this matter the implementation of our FSS heuristic uses a MATLAB function called *vpa* that gives us as many digits as we desire. In Table 3.1 we present data, our results and comparisons with nine decimal places of accuracy while the total time is reported with two decimal places expressed in seconds. The average percentage deviation from Table 3.1 is 0.012694805 whilst the average total time spent is of 4507.97 seconds which is 75.13 minutes. This suggests good accuracy when balancing quality of solution and time spent.

Figure 3.4 illustrates the result obtained by our heuristic with 50 identical circles, all centre circles with an "*o*" represent those circles expressed in the Cartesian coordinate system whilst centre circles with a $*$ are expressed in the polar system. The radius associated with all 50 circles inside the unit circle container is 0.125825489530 having 0.000000001 as the percentage deviation from the best-known.

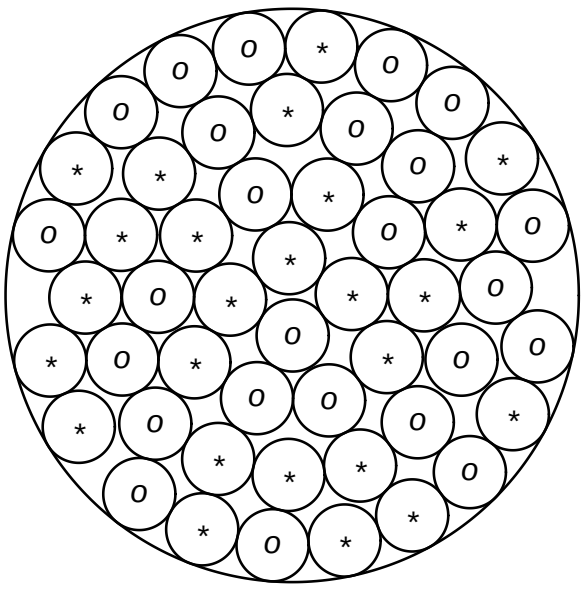


Figure 3.4: Final solution for $n = 50$ identical circles

A further experiment with the unit circle container was carried out based on results

shown in Section 4.3. The improvements presented in Table 4.4 encouraged us to run our heuristic considering instances with $2 \leq n \leq 150$ identical circles. The results showed no improvements over the best-known results, the average % deviation obtained was 0.0111 with an average total computational time of 637 seconds. Details of this experiment considering other containers can be found in Section 4.3.

3.5.2 Comparison with Mladenovic et al [62]

Mladenovic et al. [62] used an approach called reformulation descent (RD), it was applied to the packing problem for the unit circle and unit square container. Although in this section we concentrate our attention on the unit circle container, in Chapter 4 we tackle more cases with different containers, among which is the unit square. The RD method works with the Minos non-linear solver. They give results using two different formulations of the problem: one a pure Cartesian formulation (denoted by M_C), the other a pure polar formulation (denoted by M_P), both using Minos. They also give results for a pure Cartesian formulation using the Spenbar non-linear solver (denoted by SP).

Table 3.2 shows the results they obtained for the case where the container is the unit circle. That table also shows the performance of our FSS heuristic considering two non-linear solvers Snopt and Minos (for the same values of n as considered in [62]).

In Table 3.2 we give a "best-known" result as taken directly from [62]. The values seen are given as the inverse of the circle radius. The column labelled Best-known† in Table 3.2 refers to the inverse of best-known circle radius at that time and is given here precisely as in [62] with six decimal place accuracy. In computing the percentage deviation from this value the results shown for the four approaches (RD, M_C, M_P, SP) are calculated as: $100(f_{best}^{-1} - f_{bk}^{-1})/(f_{bk}^{-1})$ where f_{bk}^{-1} is the inverse of the best-known radius [so here the value used for this inverse is as shown in the table under Best-known†] and f_{best}^{-1} is the inverse of best radius found by an approach. The percentage deviations for our FSS heuristic as seen in this table are also calculated in this way.

The computation times (seconds) shown in Table 3.2 for our FSS heuristic are total computation times, that is the total time for twenty-five replications from different initial solutions (2.26GHz Intel Core 2 pc). The computation time given for the approaches in [62] is the average time per replication (over fifty replications) from different initial solutions (in seconds, 1800MHz Pentium 4 pc).

Table 3.2: Comparison with Mladenovic et al [62] for the unit circle

		% Deviation from Best-known						Total time		Average computation time as reported in [62]			
n	Best-known†	FSS Snopt	FSS Minos	RD	M_C	M_P	SP	FSS Snopt	FSS Minos	RD	M_C	M_P	SP
10	3.813026	0.00	0.00	0.00	0.00	0.00	0.00	41	94	0.00	0.02	0.01	0.29
15	4.521357	0.00	0.00	0.00	0.13	0.13	0.00	37	205	0.01	0.03	0.02	1.87
20	5.122307	0.00	0.00	0.00	0.00	0.00	0.00	47	293	0.04	0.11	0.08	5.21
25	5.752824	0.00	0.14	0.00	0.00	0.00	0.00	65	433	0.08	0.37	0.19	17.14
30	6.197741	0.00	8.30	0.00	0.00	0.00	0.00	82	516	0.16	0.52	0.29	41.69
35	6.697171	0.00	35.75	0.00	0.01	0.02	0.03	112	190	0.90	1.84	1.73	81.98
40	7.123847	0.00	41.38	0.00	0.00	0.00	0.00	109	271	1.11	2.92	1.91	179.69
45	7.572912	0.00	33.72	0.10	0.11	0.04	0.07	151	323	1.47	3.08	2.19	300.41
50	7.947515	0.00	80.46	0.06	0.03	0.00	0.02	173	444	3.19	5.16	4.41	503.78
55	8.211102	0.00	71.29	0.00	1.13	1.57	1.56	219	552	3.37	6.73	5.15	902.59
60	8.646220	0.00	85.09	0.03	0.10	0.57	0.00	246	656	4.71	7.54	6.00	1526.40
65	9.017397	0.00	68.82	0.00	0.47	0.44	0.31	289	750	16.24	12.94	10.43	2118.60
70	9.346660	-0.01	69.98	0.10	0.55	0.32	0.27	347	959	19.56	17.61	14.54	3484.63
Average ($n \leq 70$)		0.00	38.07	0.02	0.19	0.24	0.17	148	437	3.91	4.53	3.61	704.94
75	9.678344	-0.07	77.14	0.10	0.22	0.44		408	1232	26.46	22.67	17.16	
80	9.970588	-0.02	81.11	0.10	0.41	0.29		479	1366	39.15	30.99	23.62	
85	10.163112	0.00	84.51	0.72	1.43	1.10		621	1139	38.79	29.85	24.04	
90	10.546069	0.00	84.59	0.02	0.02	0.45		586	1592	96.82	47.19	47.70	
95	10.840205	0.00	84.47	0.18	0.26	0.48		693	1799	147.35	59.51	41.84	
100	11.082528	0.00	89.39	0.30	0.52	0.38		810	2063	180.32	64.96	45.02	
Average ($n \leq 100$)		-0.01	52.43	0.09	0.28	0.33	0.17	290	783	30.51	16.53	12.96	704.94

Results presented in Table 3.2 concerning the average percentage deviation of all five approaches, it is clear that our FSS heuristic has the lowest average percentage deviation (-0.01) when using the non-linear solver Snopt, this indicates that our heuristic is capable of more accurate results when compared with the other four approaches (RD, M_C, M_P, SP) presented in [62]. However using a different non-linear solver (Minos) makes comparison difficult. Summarising, our FSS approach when using Snopt produced

better quality results in less time when compared with the approaches presented in Mladenovic [62] and when using Minos as non-linear solver. In order to have a better insight of our FSS approach with these two non-linear solvers we decided to test it using single and mixed formulations, the results are presented in section 3.6.

3.5.3 Comparison with Mladenovic et al [63]

In [63] Mladenovic et al. used the formulation space search approach with a mixed formulation solving the packing problem inside the unit circle container as described in Section 2.5. They compared the percentage deviation from two approaches: RD (as in [62]) and FSS (denoted in Table 3.3 as FSS-M). The results presented were obtained on a Pentium 3, 900 MHz computer.

In Table 3.3 we present the comparisons between our results and those presented in [63]. The series of instances considered goes from $n = 50$ to 100 identical circles in steps of five. Here as before we give a "best-known" result as taken directly from [63]. The results for our FSS approach and the computational times are the same as in Table 3.2 and presented here for ease of comparison. The results presented in [63] were produced with 40 replications, the computational time given in seconds represents the average time per replication taken from [63]. For RD the results and computational time are the same as in Table 3.2.

From Table 3.3 the average percentage deviation from best-known result indicates that FSS-M improves over RD, however the average time for the FSS-M approach increase by $188.87/52.36 = 3.61$ times. In contrast, the negative average percentage deviation from our FSS approach suggest that our results are equal and in some cases better in quality than those by FSS-M and even better than the best-known presented in second column.

In FSS-M Mladenovic et al. [63] they use RD as local search, here they ignore overlap constraints for circles whose centres (at the initial solution, before optimisation) are

sufficiently far apart, however after have obtained a solution for each local search they solve the problem in an iterative way using every time a reduced formulation taken randomly from the same level set (same cardinality for C but randomly choosing the circles in C), they change from one level to another after no more improvements are found. With regards to quality of results, our FSS approach using Snopt non-linear solver outperforms FSS-M this may suggests that the strategy adopted in our FSS algorithm of using constraints that limit the range of movement for circle centres is a superior one to use. However using a different non-linear solver such as Minos makes comparison with FSS-M difficult.

Table 3.3: Comparison with Mladenovic et al [63] for the unit circle

n	Best-known†	FSS (Snopt)		FSS (Minos)		RD		FSS-M	
		% dev	Time	% dev	Time	% dev	Time		
50	7.947515	0.00	173	80.46	444	0.06	159.50	0.00	3221.60
55	8.211102	0.00	219	71.29	552	0.00	168.50	0.00	2912.40
60	8.646220	0.00	246	85.09	656	0.03	235.50	0.00	3375.60
65	9.017397	0.00	289	68.82	750	0.00	812.00	0.00	4330.00
70	9.346660	-0.01	347	69.98	959	0.10	978.00	0.01	6065.60
75	9.678344	-0.07	408	77.14	1232	0.10	1323.00	0.02	6580.40
80	9.970588	-0.02	479	81.11	1366	0.10	1957.50	0.04	9179.60
85	10.163112	0.00	621	84.51	1139	0.72	1939.50	0.18	10246.80
90	10.546069	0.00	586	84.59	1592	0.02	4841.00	0.02	11790.80
95	10.840205	0.00	693	84.47	1799	0.18	7367.50	0.07	12333.60
100	11.082528	0.00	810	89.39	2063	0.30	9016.00	0.12	13066.80
Average		-0.01	443	79.71	1141	0.15	2618.00	0.04	7554.84

3.5.4 Comparison with other work

Pushing forward a further investigation was carried out, the objective being to analyse the performance of our heuristic with other relatively recent work found in the literature. We considered four different works: Birgin and Gentil [9], Grosso et al [23], Liu et al [51] and Pintér [68]. The approach used by them has been described in Chapter 2.

Based on data taken from the respective work under consideration we calculated their corresponding percentage deviation, the average time in seconds and also presenting the non-linear solver used. This information can be seen in Table 3.4 where we also present the average percentage deviation and average time in seconds from our FSS heuristic for analogous cases produced on a Intel Core 2 pc (2.26 GHz, 4GB RAM).

Results from Table 3.4 indicates that results given by Pintér are not competitive with the other results shown.

Comparing with the Grosso et al [23] approach, it seems that our FSS approach dominates in quality of results and in computational times. For the approaches proposed in [9, 51] our approach gives higher average percentage deviation, however our results are produced in much faster time. This suggests that our FSS heuristic is a computational effective approach when balancing quality of results and computational time when compared with Birgin and Gentil [9] and Liu et al [51].

Before ending the section we need to highlight the fact that by the nature of the circle packing problem it is necessary to use a non-linear solver as part of the algorithm as done by all approaches presented in Table 3.4. But we need to be careful as we will see in Table 3.5 in Section 3.6 that results produced when changing to another solver may differ considerably even using the same algorithm. Hence, even though comparisons such as those in presented in Table 3.4 are necessary we do not know how we would had performed if those approaches had used a different non-linear solver.

Table 3.4: Comparison with other work[1]

Paper	Values of n	Average % deviation	Average time (secs)	FSS Average % deviation	FSS Average time (secs)	Solver used
Birgin and Gentil [9]	10,15,20,...,50	0	1019	1.66×10^{-7}	91	ALGENCAN
Grosso et al [23]	30,35,40,...,100	0.00059028	3784	-0.00049533	355	Snopt
Liu et al [51]	35,40,45,...,100	3.10664×10^{-10}	79870	2.00×10^{-5}	375	Snopt
Pintér [68]	10,15,20,...,60	0.59949089	195	1.36×10^{-7}	117	LGO

[1]For Birgin and Gentil [9] the computation time is for a 2.4GHz Intel Core 2 Quad, 4GB RAM pc. They start from the solution given by the heuristic algorithm of Birgin and Sorbal [10]. The time given above does not include the time for this heuristic, which [10] indicates was one hour (2GHz AMD Opteron 244 processor, 2GB RAM) for all values of n. For Grosso et al [23] the computation time is for a Pentium IV 2.4GHz, 1GB RAM pc. Liu et al [51] use five replications, but only give the time for the replication that resulted in the best solution found. To account for this we have multiplied the time for this replication by five. Their computation time is for a Pentium IV 1.6GHz, 512MB RAM pc. Pintér [68] gives results for two approaches, the results given above are for the LGO+CONCOPT approach which (effectively) dominates the other approach presented. His computation time is for an AMD Athlon 64 3200+ 2GHz pc.

3.6 Algorithmic variations

In the pursuit of a better understanding of the effect that different modifications to our heuristic may have we decided to carried out two different types of test: in Section 3.6.1 we change from a mixed formulation to single formulation, additionally we use a different solver called Minos whose results are compared with those produced by the non-linear solver Snopt. In Section 3.6.2 we investigate the possible implications that changes on specific parameters of a mixed formulation such as the number of circles in set C (consequently in P) have on the results presented in Tables 3.1- 3.4, simultaneously we investigate different ways to update sets C and P.

3.6.1 Single formulations

We investigate the contribution of switching between formulations and the use of a different solver in our formulation space search heuristic. We carried out a test to evaluate performance, the test is divided in two and involves using single formulations of the packing problem. In first part of the test the single formulation used is for all circles to be expressed in the Cartesian coordinate system, technically that corresponds to $C = \{1, ..., n\}$ and $P = \emptyset$ (with no switching of the sets C and P in Algorithm 3.5). In a similar manner the second part of the test considers a polar formulation only, where $P = \{1, ..., n\}$ and $C = \emptyset$ (with no switching of the sets C and P in Algorithm 3.5).

The non-linear solver used for this test is Minos which was used in [62, 63]. Table 3.5 presents the average percentage deviation calculated with results from the test comparing single formulations using Minos and Snopt as the non-linear solvers. The Best known results are those reported in Packomania in January 2011. The instances considered go from $n = 10, 15, ..., 100$ identical circles.

Table 3.5 shows that with single and mixed formulations, non-linear solver Minos produced higher average % deviation when compared with Snopt.

Table 3.5: Results for the unit circle comparing a mixed formulation with our FSS heuristic with single formulations using Minos and Snopt

	Packomania	Cartesian Formulation				Polar Formulation				Mixed Formulation			
n	Best known	Minos	time	Snopt	time	Minos	time	Snopt	time	Minos	time	Snopt	time
10	0.262259	0.000000	90	0.000000	27	0.000000	99	0.000000	28	0.000000	94	0.000000	41
15	0.221173	0.000000	137	0.000000	34	0.129917	155	0.000000	37	0.000000	205	0.000000	37
20	0.195224	0.000000	197	0.000000	41	0.000000	162	0.000000	43	0.000000	293	0.000000	47
25	0.173828	0.000000	272	0.000000	58	0.412662	334	0.000000	65	0.139224	433	0.000000	65
30	0.161349	0.000000	304	0.000000	73	7.428389	597	0.000000	86	8.300765	516	0.000000	82
35	0.149317	0.000000	508	0.000000	103	4.579150	865	0.000020	115	46.211209	190	0.000000	112
40	0.140374	0.000000	585	0.000000	101	34.304136	957	0.000000	114	41.378935	271	0.000000	109
45	0.132050	0.077918	878	0.000000	135	31.705839	1372	0.000106	152	33.722255	323	0.000000	151
50	0.125825	0.000000	1216	0.000000	163	32.147063	1898	0.000000	185	80.458204	444	0.000000	173
55	0.121786	1.462660	1300	0.000000	212	51.168021	2326	0.000000	241	71.290774	552	0.000000	219
60	0.115657	0.000000	1739	0.000000	237	53.011893	2902	0.000000	251	85.090458	656	0.000000	246
65	0.110897	0.000000	2212	0.000000	292	56.149072	3655	0.000000	314	68.820562	750	0.000000	289
70	0.107002	0.231772	2814	0.000000	343	64.670124	4647	0.000701	391	69.976156	959	0.000000	347
75	0.103391	0.028387	3410	0.023851	395	69.555199	5971	0.000077	429	77.140604	1232	0.000010	408
80	0.100320	0.116767	3936	0.016567	468	70.754200	7409	0.000010	531	81.109675	1366	0.000030	479
85	0.098395	1.152741	4638	0.000000	581	77.951434	8812	0.000000	659	84.513237	1139	0.000000	621
90	0.094822	0.195925	5221	0.000011	587	73.146259	9988	0.000000	646	84.589831	1592	0.000000	586
95	0.092249	0.407115	5721	0.029756	696	79.502268	11524	0.000813	726	84.472057	1799	0.000065	693
100	0.090235	1.038331	6477	0.001718	801	80.424956	13176	0.005441	835	89.388022	2063	0.000177	810
Average		0.247980	2192	0.003784	281	41.423189	4045	0.000377	308	52.979051	783	0.000015	290

Average % deviation from single formulations using Minos showed a radical difference, in our experience this is due to the fact that although we provide random initial solutions at every iteration in the implementation of our FSS approach, the solver Minos tends to prefer as solution lower/upper limits of the variables, whilst for the Cartesian system the limits are $x_i \in [X_i - \Delta, X_i + \Delta]$ and $y_i \in [Y_i - \Delta, Y_i + \Delta]$ for the polar system the limits for variables are $r_i \in [0, 1]$ and $\theta_i \in [0, 2\pi]$ as stated in equations (3.7)-(3.10) and so, variables r_i with solution at the upper bound, during correction step are randomly re-allocated inside the unit circle therefore the maximum radius has to be reduced to guarantee no overlaps.

In summary based on results presented in Table 3.5 we can say that non-linear solver Minos is not effective when compared with Snopt. Regarding single formulations it is clear that the non-linear solver Snopt gives lower percentage deviations. Also, results from Table 3.5 and Table 3.1 show evidence that much lower percentage of deviation is obtained if we use a mixed formulation.

Another approach that works with equivalent formulation can be seen in PhD thesis [70], here the author applies variable neighbourhood search to solve the circle packing problem switching between two single formulations: first formulation is one aiming for the maximum common radii for n identical circles to be packed inside the unit circle container, whilst the second formulation aims to minimise the size of the container considering unit circles to be packed. They also solve the case for the square container in a similar fashion.

3.6.2 Alternative strategies

The results produced by our FSS heuristic are based on particular fixed settings, hence if we do even slight modifications to our algorithm it may lead to different results. In addition to this we know that when working with non-linear optimisation problems one of the concerns that may lead to better results is how we set the initial solution. Therefore

in this section we present the strategies that we considered to investigate the performance of our FSS heuristic under some modifications related to the initial solutions.

The strategies considered take into account two elements of variations: the initial number of circles in set C whose centres are expressed in Cartesian coordinate system, consequently in set P, whose centres are in polar coordinate system (as $|P| = n \setminus |C|$) and the systematic fashion adopted to select and update through the algorithm those circles being in set C or P.

To get some understanding about the influence that the strategies have we decided to initially set the number of circles in set C as m ($|C| = m$) where parameter m is defined as $m = \frac{n}{3}, \frac{n}{4}, \frac{n}{5}, \frac{2n}{3}, \frac{2n}{4}, \frac{2n}{5}, \frac{3n}{4}, \frac{3n}{5}, \frac{4n}{5}$. For computational reasons the investigation carried out was with values for $n = 10, 15, ..., 100$ identical circles.

Strategy 1 is called ***fix and switch FS****(m)*. It consists of fixing the initial cardinality of C to m, where we start by randomly allocating m circles to C. At each iteration we update C and P by switching them ($C \leftarrow P$, $P \leftarrow \{1, ..., n\} \setminus C$). In this strategy the cardinality of C will alternate, first m, then $n - m$, then m, etc. A similar alternative is presented in [63], in their approach the value of parameter m changes when the solution cannot be improved and the number of circles in the Cartesian system is increased in a systematic fashion.

Strategy 2 is called ***fix and random FS****(m)*. It consists of fixing the initial cardinality of C to m, where we start by randomly allocating m circles to C. At each iteration we update C and P by randomly choosing which circles will be allocated in C. In this strategy the cardinality of C will always be m.

Strategy 3 is called ***switch outer SO****(m)*. It consists of fixing the initial cardinality of C to m, where we start by randomly allocating m circles to C. At each iteration we update C and P by switching only those circles i whose distance from the origin $(0, 0)$ of the unit circle to their centre is greater than 0.75 (i.e. $\sqrt{x_i^2 + y_i^2} \geq 0.75$). In this strategy the cardinality of C will vary. Here we used 0.75 as it divides the unit circle into two

portions of (approximately) equal area, an inner area where the centre coordinates are left unchanged at each iteration and an outer area where they are switched.

Whilst there are (potentially) a large number of possible strategies that can be examined the three considered here seemed to us to capture different elements:

- in *fix and switch FS(m)* we have a fixed alternating cardinality and circles swap between Cartesian and polar coordinates
- in *fix and random FR(m)* we have a fixed cardinality for C, but circles are randomly allocated to C at each iteration
- in *switch outer SO(m)* the cardinality of C is initially considered fixed, however circles with centre close to the origin never change formulation, whilst circles whose centre is outside of a circle with radius 0.75 always change formulation (and hence C has variable cardinality).

Note here that the results presented above in Tables 3.1-3.4 are for a fix and switch strategy FS(M), but with M generated by randomly allocating each circle either to C or to P (so M will be randomly distributed around $n/2$). With three basic strategies, each with nine values for m ($\frac{n}{3}$, $\frac{n}{4}$, $\frac{n}{5}$, $\frac{2n}{3}$, $\frac{n}{2}$, $\frac{2n}{5}$, $\frac{3n}{4}$, $\frac{3n}{5}$, $\frac{4n}{5}$), we have 27 different strategies to compare with our existing strategy FS(M).

As these strategies, e.g. FR(m), may involve the use of random numbers we adopted a statistical hypothesis testing approach to comparing them against our existing strategy in order to judge whether, or not, there was sufficient statistical evidence to conclude that an alternative strategy was better than our existing strategy.

In our hypothesis testing the null hypothesis was:

H_0: the average percentage deviation from our current strategy is equal to the average percentage deviation given by a specific alternative strategy

The alternative hypothesis was:

H_1: the average percentage deviation from our current strategy is greater than the

average percentage deviation given by a specific alternative strategy (equivalently that the alternative strategy gives a lower percentage deviation than our current strategy) This is a one-sided hypothesis test. If we reject H_0 (accept H_1) then we (statistically) have evidence that there exists a better strategy than our current strategy.

We could use a paired t-test here, but since that technically assumes normality, we chose to use a Wilcoxon signed rank test. We computed the p-values for our one-sided hypothesis test (using the R language for statistical computing). For those unfamiliar with hypothesis testing the p-value is the probability that the result we observe would occur by chance if the null hypothesis was true. If the p-value is small then this provides evidence that the null hypothesis H_0 is not true and so the alternative hypothesis H_1 should be accepted. In judging whether the p-value is small the standard approach is to compare against values such as 0.05 and 0.01 (significance levels of 5% and 1% respectively).

Table 3.6: p-values for the comparisons between the 27 described strategies and current strategy adopted $[FS(M)]$ in our heuristic

	$n/2$	$n/3$	$n/4$	$2n/3$	$2n/4$	$2n/5$	$3n/4$	$3n/5$	$4n/5$
$FS(m)$	0.9350	0.8515	0.9044	0.9650	0.9455	0.9518	0.9044	0.9422	0.6569
$FR(m)$	0.8646	0.9044	0.9093	0.8940	0.9752	0.9650	0.9229	0.9487	0.8376
$SO(m)$	0.8646	0.8709	0.9487	0.9892	0.9350	0.9386	0.9229	0.8376	0.9093

In Table 3.6 we show the p-values from the hypothesis test contrasting the 27 strategies presented above with current strategy adopted ($FS(M)$) in our heuristic. We observed that for each of our 27 different strategies the associated p-value was always much greater than 0.05, indicating that the null hypothesis should be accepted at the 5% significance level. Based on this we would conclude that there is no evidence that our current strategy is out-performed by any of the 27 alternative strategies we examined.

Of course we are aware that we could have investigated further here. For example

we could have examined further values for m in fix and switch FS(m), fix and random FR(m) and swap outer SO(m). We would simply comment here that readers familiar with algorithmic work such as we present here will be aware that there are always different choices to be made, and it is impossible to examine all alternatives in detail. However we believe that the quality of the results seen in Tables 3.1-3.4, plus the quality of our FSS heuristic when considered against individual papers in the existing literature, mean that the fix and switch FS(M) strategy adopted is justified.

3.6.3 Setting numerical factors

The numerical settings that form the framework of our heuristic are: Δ factor, the number of iterations and the number of replications adopted.

Δ factor. The Δ factor is used to create a range of movement (a square vicinity) around a known point denoted as X_i for every circle i with three different purposes:

- to determine if a pair (i, j) of circles is an element of the Q set as described in Section 3.3.1
- to determine a new random solution for next iteration
- it forms part of the limits of a subset of constraints imposed only for circles expressed in the Cartesian system given in equations (3.7) and (3.8)

The final numerical factor $\frac{2}{3}$ associated to Δ was determined after conducting several experiments with five instances with $n = 25, 51, 77, 103$ and 128 identical circles. In order to get insight into what would be an appropriate numerical value for Δ and considering that we were in the first stage of the heuristic, we arbitrarily decided to conduct the experiments running our heuristic for 80 iterations. For every experiment we considered a different value for Δ taken from the succession of numbers: $0.2, 0.25, ..., 1.00, 1.25, 1.50$. To determine the most suitable numerical value for Δ we calculated the average percentage deviation for each value tested from the previous list considering the best solution

obtained with the five instances, Figure 3.5 shows the variations on the average percentage deviation with respect to each Δ factor associated. From Figure 3.5 we can see that

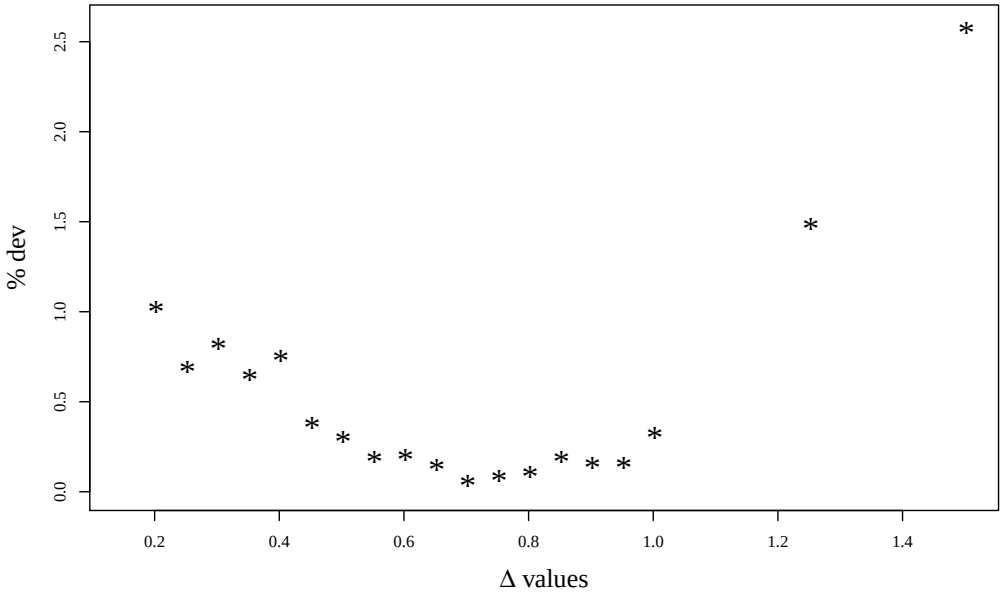


Figure 3.5: Variations in % deviation depending on Δ values

best factors for Δ (those with % deviation closer to zero) are between 0.65 and 0.80, hence we ran our heuristic to get deeper into the analysis, having the same five instances as before but this time we set the number of replications to five, each replication consisting of 80 iterations as before. In Table 3.7 we report the data resulting from these experiments: possible Δ values and their respective percentage deviation. These results suggest that the best factor to adopt is $\frac{2}{3}$.

Number of iterations. Deciding how many times we should run our heuristic requires some considerations: having to many iterations may be reflected in more computational time and it may be not contribute to the improvement of the current best solution, too few iterations may gives us fast solutions possibly with a short exploration of the set of feasible solutions. Hence, we conducted a test considering the same five instances as before ($n = 25, 51, 77, 103$ and 128 identical circles), we ran the algorithm for

Table 3.7: Values for Δ

Δ factors	% deviation
.65	0.073977017
2/3	0.011964561
.70	0.41481330
.75	0.038537852
.80	0.038463637

100 iterations recording the minimum percentage deviation and the average percentage deviation after $25, 50, 75$ and 100 iteration limits. Table 3.8 presents the results from the test conducted and it is presented in columns: the number of limit iterations, the minimum percentage deviation, the average up the respective limit iteration and the total time spent shown in minutes. Considering the minimum average percentage deviation we

Table 3.8: Iterations

Number of iterations	Minimum % deviation	Average % deviation	Total time in minutes
25	0.168361745	1.597598967	48.82
50	0.066425336	0.847013129	98.20
75	0.054332132	0.585499147	147.27
100	0.018546917	0.447976901	196.22

clearly see that the more iterations we consider the bigger the chances to obtain a better quality solution. Regarding the total time spent, looking at the tendency from all limit iterations we may say that roughly every iteration takes around two minutes (48.82/25 = 1.9528), in order to gain more accuracy and balancing time spent we decided to set the number of iteration between 75 and 100 (in fact 80 iterations). The information provided in this test allows to roughly predict that any solution obtained after 80 iterations would have a minimum percentage deviation laying in the range (0.018546917, 0.054332132) approximately, and an average percentage of deviation within a range (0.447976901,

0.585499147) with total time between 147.27 to 196.22 minutes.

Number of replications. We call replications the repeated action to run the algorithm for certain number of times, as in our heuristic the initial solution is randomly chosen, hence replicating provides a systematic random fashion to explore the solution space. In order to determine the number of replications for our algorithm we ran our heuristic with previous five instances of $n = 25, 51, 77, 103$ and 128 identical circles for 15, 20, 25, 50, 75 and 100 replications. Table 3.9 shows the minimum average percentage deviation after k replications, the average percentage deviation for k replications and the total time spent reported in minutes for the five instances. Results from Table 3.9

Table 3.9: Replications

Number of replications (k)	Minimum % deviation	Average % deviation	Total time in minutes
15	0.026667205	0.057170468	28.81
20	0.026667205	0.049544652	39.90
25	0.019892933	0.043787735	49.90
50	0.017288440	0.030628824	99.80
75	0.008886488	0.024613665	148.88
100	0.008886487	0.020681870	194.55

indicate that the lowest % deviation 0.008886488 is obtained with 75 replications, with an average % deviation 0.024613665 in 148.88 total minutes (2 hours 28 minutes). Here balancing accuracy of solution with computational time spent we set 25 as the number of replications for our heuristic. This means that a solution obtained after 25 replications may have on average a % deviation 0.043787735 in 49.9 minutes.

3.7 Formulation space and search space

Formulation space search combines the exploration of two different spaces: the formulation space and the search space. The formulation space is defined as the space

that considers all different (but valid) formulations of a given problem, as a space it induces a distance between any two given formulations. Mladenovic et al [62] define the distance between two formulations in the context of circle packing as the number of circles that differ in both formulations. To clarify let us consider two formulations denoted as ϕ_1 and ϕ_2 respectively, hence the distance between ϕ_1 and ϕ_2 is given by: $d(\phi_1, \phi_2) = |\{C_{\phi_1} \cup C_{\phi_2}\} \setminus \{C_{\phi_1} \cap C_{\phi_2}\}|$ examples are given below. With regards of the search space is what we often call as the space of feasible solutions, in this context we refer to the geometry of the container, here the unit circle.

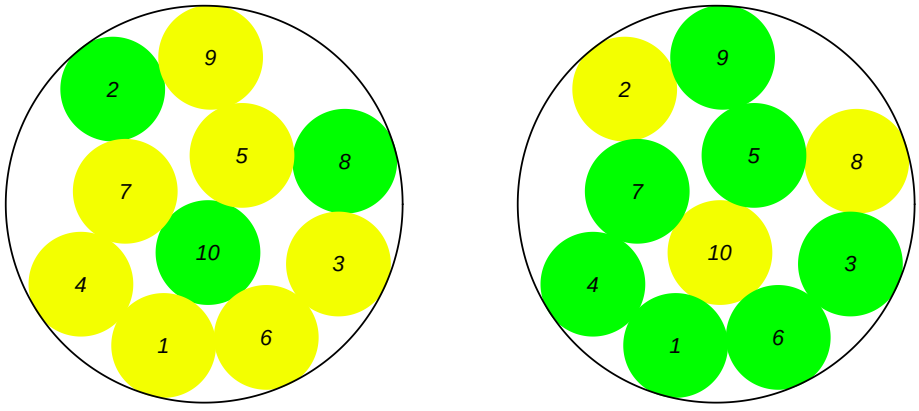
3.7.1 Reformulation descent and formulation space search

Reformulation descent and formulation space search have common grounds, both use different (but valid) formulations of the original problem to obtain a better solution. However whilst a solution obtained using reformulation descent is a solution that is a stationary point for all the proposed formulations, a solution obtained with formulation space search is one that is the best solution found after exploring the formulations considered. Above in our final FSS heuristic presented in algorithm 3.5 we explored several formulations which have been detailed in section 3.6.2. Based on the statistical tests carried out in section 3.6.2 we decided to use two mixed formulations ϕ_1 and ϕ_2 where $C_{\phi_1} = P_{\phi_2}$ (consequently $P_{\phi_1} = C_{\phi_2}$), according to the definition of distance between two formulations we can say that for our two formulations the distance is n ($d(\phi_1, \phi_2) = n$). In order to diversify the initial solution at every iteration we explored the nearby space by creating a new random initial solution close to previous one in a range of movement Δ.

Tested formulations and the distance between them Before the final version of our heuristic algorithm was set, we tested three main strategies detailed in section 3.6.2, for each strategy we set $|C| = m$ where parameter m is defined as $m = \frac{n}{3}, \frac{n}{4}, \frac{n}{5}, \frac{2n}{3}, \frac{2n}{4}, \frac{2n}{5}, \frac{3n}{4}, \frac{3n}{5}, \frac{4n}{5}$.

In order to illustrate the distance between any two formulations with each strategy, let us consider for all cases $n = 10$ circles and that the parameter m is given by $\frac{n}{3}$ hence, in all cases we have $m = \frac{10}{3}$ which is rounded to 3.

Let us recall that Strategy 1 starts by randomly allocating 3 circles into C and the remaining circles $(n - m = 7)$ are allocated into P. In order to have a clear idea let us consider formulation 1 ϕ_1 with $C_{\phi_1} = \{8, 2, 10\}$, therefore $P_{\phi_1} = \{7, 4, 3, 6, 9, 5, 1\}$. The solution obtained by the non-linear solver is depicted in Figure 3.6(a) where green circles are expressed in the Cartesian coordinates system whilst the yellow circles are expressed in the polar system. For the next iteration we swap to formulation 2 (ϕ_2) where $C_{\phi_2} = P_{\phi_1}$ and $P_{\phi_2} = C_{\phi_1}$. Let us recall from section 3.3.2 that the initial solution in formulation 2 is randomly generated at a Δ distance to the centre of the circles in the current solution which are considered known points. Hence, in Figure 3.6(b) we only show the change of the formulation. With regard to the distance between these two formulations by following the definition we can see that it is 10. As this is the behaviour of the nine cases considered in strategy 1, we can say that we always work with two formulations and the distance between the two is given by the number of circles n. ***Hence for this strategy we have a fixed distance between formulations.***

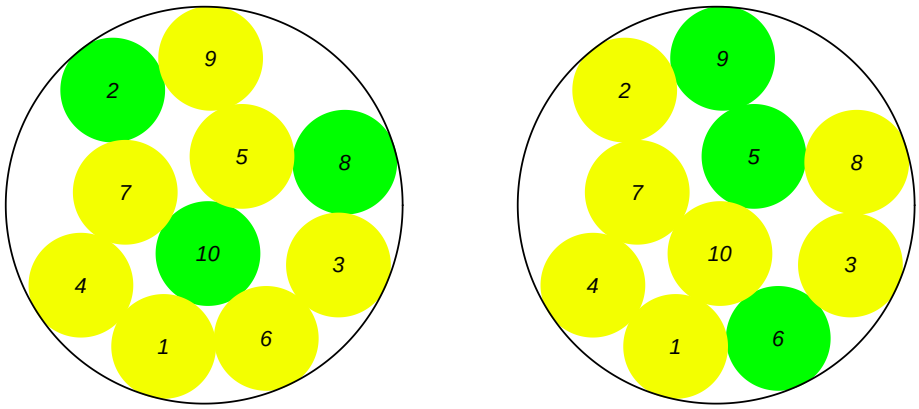


(a) Solver solution with formulation ϕ_1

(b) Current solution showing the swap to formulation ϕ_2

Figure 3.6: Example of Strategy 1 change of formulation with $n = 10$ circles

Strategy 2 consists of fixing the number of circles in C as m, for the initial solution we randomly allocate m circles to C then after the solver has given a solution we start the next iteration with a new initial solution that randomly allocates m circles to C. Considering the previous example in Figure 3.7(a) we have $C_{\phi_1} = \{8, 2, 10\}$ and $P_{\phi_1} = \{7, 4, 3, 6, 9, 5, 1\}$, however in next iteration the circles are randomly re-allocated, let us say that the re-allocation is set to $C_{\phi_2} = \{5, 6, 9\}$ and $P_{\phi_2} = \{1, 4, 2, 10, 8, 3, 7\}$, then in Figure 3.7(a) we show the swap from formulation ϕ_1 to ϕ_2 of current solution. The distance between these two formulations is $d(\phi_1, \phi_2) = |\{8, 2, 10, 5, 6, 9\} \setminus \emptyset| = 6$. As the elements in C may change at every iteration, this means that we are considering at most 80 different formulations per replication, that makes a total of 2000 possible different formulations for all 25 replications, the distance between any two given formulations might vary as they are dependent on which circles are allocated in C. ***Hence for this strategy we have a variable distance between formulations***.



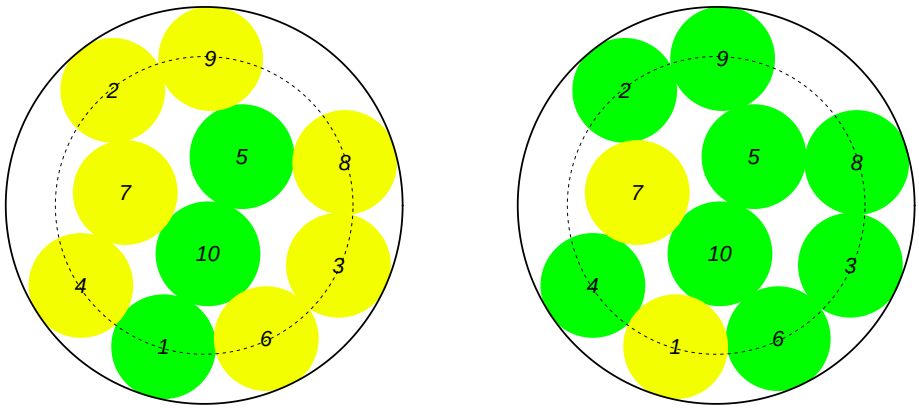
(a) Solver solution with formulation ϕ_1

(b) Current solution showing the swap to formulation ϕ_2

Figure 3.7: Example of Strategy 2 change of formulation with $n = 10$ circles

Strategy 3 considers a random initial solution setting $|C| = m$ for iteration one, then after we have obtained a solution from the non-linear solver we will swap only those circles that satisfy $x_i^2 + y_i^2 \geq 0.75$ (to be close to the edge of the unit circle

container), this implies that although circles close to the origin will always be in the same set as they were in at iteration one, the cardinality of C will vary depending on those circles that are close to edge of the container. To illustrate how strategy 3 works let us consider initially $C_{\phi_1} = \{1, 5, 10\}$ and $P_{\phi_1} = \{2, 3, 4, 6, 7, 8, 9\}$, after we obtained a solution from the non-linear solver we only swap circles whose distance from the origin of the Cartesian plane to their centre is greater than 0.75, in this case looking at Figure 3.8(a) this constraint is depicted with a dotted circle dividing those circles that will swap from C to P and vice-versa. As we can see, circles $\{5, 7, 10\}$ are close to the origin hence, they will remain in the set where they were initially placed, and circles $\{1, 2, 3, 4, 6, 8, 9\}$ are around the edge of the container, therefore in next iteration we consider formulation ϕ_2 with $C_{\phi_2} = \{2, 3, 4, 5, 6, 8, 9, 10\}$ and $P_{\phi_2} = \{1, 7\}$ as shown in Figure 3.8(b). In this case the distance between the two formulations is $d(\phi_1, \phi_2) = |\{1, 2, 3, 4, 5, 6, 8, 9, 10\} \setminus \{5, 10\}| = 7$. In general we can conclude that ***for this strategy we have a variable distance between formulations.***



(a) Solver solution with formulation ϕ_1

(b) Current solution showing the swap to formulation ϕ_2

Figure 3.8: Example of Strategy 3 change of formulation with $n = 10$ circles

3.8 Conclusions

In this chapter we addressed the circle packing problem of identical circles inside the unit circle container. We presented our heuristic algorithm, which is based on the Formulation Space Search method. We described the procedures implemented in our FSS heuristic to ensure a feasible and accurate solution. We presented the final pseudocode which was accompanied by some pictures detailing the steps in the iterations considered. We gave computational results, we provided a description of the tests carried out in order to generate the framework of our current FSS heuristic for the case with identical circles, here is worth highlighting the role of Δ factor inside the algorithm, it is used:

- to determine if a pair (i, j) of circles is an element of the Q set
- to determine a new random solution for next iteration
- it also sets the limits of a subset of constraints imposed only for circles expressed in the Cartesian system.

We also presented an extensive analysis comparing our results with others found in the literature and others produced with alternative strategies. With respect to the comparisons made we can conclude that our approach:

- dominates in quality of solution when considering papers based on the formulation space search approach (Mladenovic, et al [62] and Mladenovic, et al [63]), as presented in section 3.5.2 and section 3.5.3;
- is an effective approach when balancing quality of solution and computational time when considering results from Birgin [9] and Liu [51] as presented in section 3.5.4;
- produced poor results when evaluating the polar single formulation in conjunction with a different solver (Minos) as presented in table 3.5 in section 3.6.1;

- with the current strategy (based on the hypothesis testing) none of the 27 alternative strategies considered was superior, as presented in section 3.6.2 .

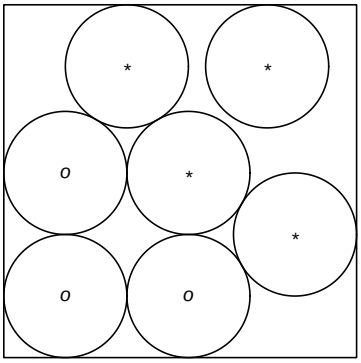
Chapter 4

Packing identical circles inside different containers

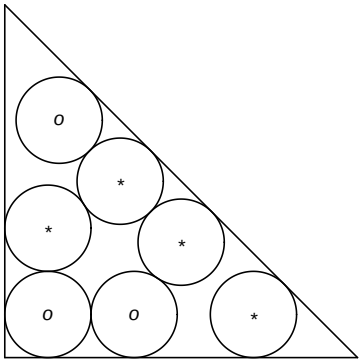
Containers with other shapes have been considered in the literature as we mentioned in Section 2.3.3 mainly comprising rectangular containers and different types of triangles. However, here we present container shapes such as a semicircle and a circular quadrant that, although they are considered in Packomania website [72], have not received much attention in the literature (to the best of our knowledge). The six different shaped containers considered are: the unit square, two rectangles of different dimension ($L = 5$ and $L = 10, W = 1$), a right angled isosceles triangle, a semicircle and a circular quadrant. Figure 4.1 illustrates the six different containers that have been considered here with $n = 7$ identical circles for illustrative purposes.

The aim of this chapter is to demonstrate the capabilities of our FSS heuristic for solving the packing problem with identical circles inside different shaped containers. Hence we focus our attention on the modifications to the mathematical model given in equations (3.1)-(3.11) and to the heuristic algorithm 3.5, for ease of reference in Section 4.1 we repeat them both. In Section 4.2 we detail the modifications needed for each container. In Section 4.3 we give computational results and in Section 4.4 we finish

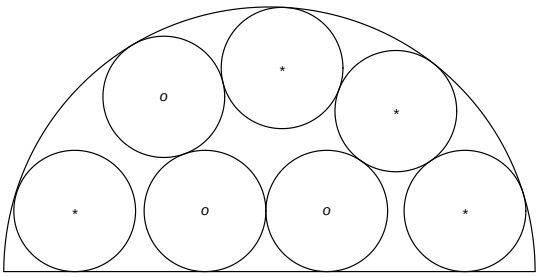
the chapter with some conclusions.



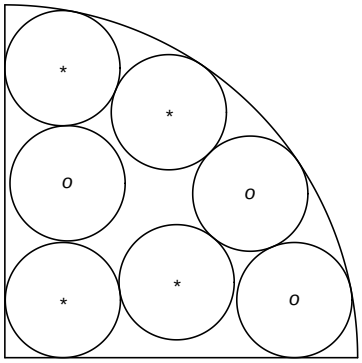
(a) Unit square



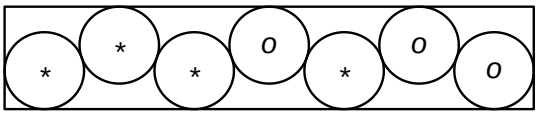
(b) Right angled isosceles triangle



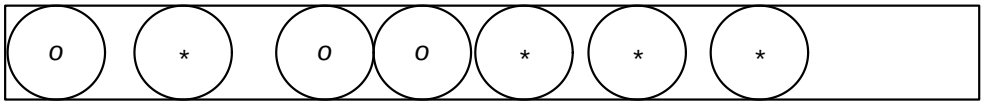
(c) Semicircle



(d) Circular quadrant



(e) Rectangle 5×1



(f) Rectangle 10×1

Figure 4.1: Example of different containers with 7 identical circles

4.1 Model and algorithm for the packing problem

As we mentioned at the beginning of this chapter, here for ease of reference we give again the mathematical model for the packing problem with identical circles with the unit circle container, we have also again included the heuristic algorithm developed. The model is given in equations (4.1)-(4.11), whilst our heuristic algorithm is given here again in the pseudo-code 4.6.

$$\max \quad R \tag{4.1}$$

$$st$$

$$x_i^2 + y_i^2 \leq (1-R)^2 \qquad \forall i \in C \tag{4.2}$$

$$r_i \leq 1 - R \qquad \forall i \in P \tag{4.3}$$

$$(x_i - x_j)^2 + (y_i - y_j)^2 \geq 4R^2 \qquad \forall (i,j) \in Q \text{ with } i,j \in C\ i < j \tag{4.4}$$

$$(x_i - r_j\cos(\theta_j))^2 + (y_i - r_j\sin(\theta_j))^2 \geq 4R^2 \qquad \forall (i,j) \in Q \text{ with } i \in C\ j \in P \tag{4.5}$$

$$r_i^2 + r_j^2 - 2r_ir_j\cos(\theta_i - \theta_j) \geq 4R^2 \qquad \forall (i,j) \in Q \text{ with } i,j \in P\ i < j \tag{4.6}$$

$$X_i - \Delta \leq x_i \leq X_i + \Delta \qquad \forall i \in C \tag{4.7}$$

$$Y_i - \Delta \leq y_i \leq Y_i + \Delta \qquad \forall i \in C \tag{4.8}$$

$$0 \leq r_i \leq 1 \qquad \forall i \in P \tag{4.9}$$

$$0 \leq \theta_i \leq 2\pi \qquad \forall i \in P \tag{4.10}$$

$$0 \leq R \leq R_{overlap} \tag{4.11}$$

Let us briefly recall that equation (4.1), maximises the radius associated with the circles. Equation (4.2) are the constraints which ensure that every circle is fully inside the container, in this case the unit circle. Its polar equivalent is given in equation (4.3). Equations (4.4)-(4.6) ensure that no circles overlap each other. Equations (4.5)-(4.6) are as equation (4.4), but where one or both of the circles has its centre expressed in polar coordinates. Equations (4.7)-(4.11) are the limits on the variables.

Algorithm 4.6 Formulation space search pseudocode

Function $(R_{best}, X_{best}, Y_{best}) \leftarrow FSS(n, replication\_limit, iteration\_limit)$

Initialisation: $|C| \leftarrow \lfloor n/2 \rfloor$ $\Delta \leftarrow \frac{2}{3}R_{overlap}$ $R_{best} \leftarrow 0$ $t\_rep \leftarrow 0$

repeat

$t \leftarrow 0$

$(x^0, y^0, X, Y) \leftarrow InitialSolution(n, |C|)$ {a first initial solution}

repeat

$Q \leftarrow OverlapSet(X, Y, \Delta, R_{overlap})$ {find the overlap set Q}

$(x, y, R) \leftarrow NLP(x^0, y^0, C, P, Q, X, Y, \Delta, R_{overlap})$

$(R^*, x, y) \leftarrow Correction(x^0, y^0, x, y)$ {correct the radius}

$R_{best} \leftarrow \max\{R_{best}, R^*\}$ {update R_{best}}

$(X_{best}, Y_{best}) \leftarrow (x, y)$ {save coordinates associated with R_{best}}

if $R^* \leq 0.001$ **then**

$\Delta \leftarrow \frac{1}{10}R_{overlap}$

else

$\Delta \leftarrow \frac{2}{3}R^*$ {update Δ}

end if

$t \leftarrow t + 1$ {update iteration counter}

$(X, Y) \leftarrow (x, y)$ {set (X, Y) to the current solution}

$x^0 \in (X - \Delta, X + \Delta)$ and $y^0 \in (Y - \Delta, Y + \Delta)$ {new initial solution (x^0, y^0)}

$C \leftarrow P$ $P \leftarrow \{1, ..., n\} \setminus C$ {switch the sets C and P}

until $t = iteration\_limit$

$t\_rep \leftarrow t\_rep + 1$ {update replication counter}

until $t\_rep = replication\_limit$

As the modifications to the heuristic are related to the correction procedure, let us recall that this procedure modifies the solution given by the solver (if needed) reducing

as much as possible the size of the radii to avoid any overlaps between the circles (equation (4.12)), and between any circle with the container (equation (4.13) considers the unit circle container). Details about the model and/or the algorithm can be found in Chapter 3.

$$R = \min\left\{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}/2 \mid i = 1, ..., n;\ j = 1, ..., n;\ i < j\right\} \tag{4.12}$$

$$R^* = \min\left\{R, \min\left\{1 - \sqrt{x_i^2 + y_i^2} \mid i = 1, ..., n\right\}\right\} \tag{4.13}$$

4.2 Different shaped containers

Six different shaped containers were investigated, in Section 4.2.1 we present the appropriate modifications for the rectangular shaped containers considered: the unit square and two rectangles with different dimensions (lengths 5 and 10, width 1). Another three shaped containers are considered: Section 4.2.2 presents the modifications for the right-angled isosceles triangle, Section 4.2.3 presents those for the semicircle and Section 4.2.4 for the circular quadrant.

4.2.1 Rectangular and square containers

For the case of a rectangular container we consider a rectangle of length L, width W, centred at the Cartesian origin. If the container is the square then (obviously) $L = W$, where the case of the unit square corresponds to $L = W = 1$. The modifications needed to change from the container being the unit circle (as considered in Chapter 3) are relatively minor and we detail them below.

4.2.1.1 Formulation modifications

Regarding formulations changes we now define $R_{overlap}$, the upper bound on the radius, using $n\pi R_{overlap}^2 = LW$, so $R_{overlap} = \sqrt{LW/n\pi}$.

Equations (4.2) and (4.3) that ensure that the circles are inside the container must be changed. Equation (4.2) is replaced by:

$$-L/2 \leq x_i + R \leq L/2 \qquad -W/2 \leq y_i + R \leq W/2 \qquad \forall i \in C \tag{4.14}$$
$$-L/2 \leq x_i - R \leq L/2 \qquad -W/2 \leq y_i - R \leq W/2$$

The polar replacement of equation (4.3) is simply as equation (4.14), but with Cartesian variables replaced by their polar equivalents.

With respect to the constraints, equations (4.7)-(4.11), setting variable limits then in the rectangular case equations (4.7) and (4.8) are no longer valid and are deleted. Equation (4.9), which bounds the polar radius, is replaced by:

$$0 \leq r_i \leq \sqrt{(L/2)^2 + (W/2)^2} \qquad \forall i \in P \tag{4.15}$$

Note here that for this container, as for the other containers considered below, very few changes are needed from the unit circle case considered in Chapter 3. In other words, the formulation adopted and the FSS heuristic are easily extendible to different containers.

4.2.1.2 FSS heuristic modifications

For the heuristic algorithm 4.6 the changes concern the correction procedure, namely $Correction(x^0, y^0, x, y)$ and finding the radius that corresponds to the given circle centres such that the circles do not overlap and are fully inside the container. Equation (4.12), which ensures that the circles do not overlap, is as before. Equation (4.13), to ensure that the circles are fully inside the container, is replaced by

$$R^* = \min\{R, \min\{L/2 - |x_i|, W/2 - |y_i| \mid i = 1, ..., n\}\} \tag{4.16}$$

4.2.2 Triangular container

For the case of the triangular container in Section 2.3.3 we made reference to some works found in the literature [19, 58–60] that mainly focus on equilateral triangles. Here we

consider an right-angled isosceles triangle (as in Specht [72]) where the two equal sides are each of length L and they join at the origin of the Cartesian plane. Their intersections with the third side of the triangle are at coordinates $(0, L)$ and $(L, 0)$.

4.2.2.1 Formulation modifications

For the heuristic algorithm 4.6 changes, $R_{overlap}$, the upper bound on the radius, is now defined using $n\pi R^2_{overlap} = L^2/2$, so $R_{overlap} = L/\sqrt{2n\pi}$.

Equations (4.2) and (4.3) that ensure that the circles are inside the container must be changed. Equation (4.2) is replaced by:

$$x_i + y_i + \sqrt{2}R \leq L \qquad \forall i \in C \tag{4.17}$$

Equation (4.3) is replaced by the equivalent polar expression of (4.17). Additionally we need to include constraints (4.18) to ensure that the circles are fully inside the triangle container. As before the polar equivalent of constraints (4.18) is also included.

$$x_i \geq R \qquad y_i \geq R \qquad \forall i \in C \tag{4.18}$$

With respect to the constraints, equations (4.7)-(4.11), setting variable limits then equations (4.7) and (4.8) are no longer valid and are deleted. Equations (4.9) and (4.10) are replaced by

$$0 \leq r_i \leq L \qquad 0 \leq \theta_i \leq \pi/2 \qquad \forall i \in P \tag{4.19}$$

4.2.2.2 FSS heuristic modifications

Equation (4.13), to ensure that the circles are fully inside the container, is replaced by

$$R^* = \min\left\{R, \min\left\{x_i, y_i, \{(L - x_i - y_i)/\sqrt{2}\} \mid i = 1, ..., n\right\}\right\} \tag{4.20}$$

4.2.3 Semicircular container

Here we consider the semicircular container as the upper half of the unit circle centred at the origin of the Cartesian plane as in Specht [72].

4.2.3.1 Formulation modifications

$R_{overlap}$, the upper bound on the radius, is now defined using $n\pi R^2_{overlap} = \pi 1^2/2$, so $R_{overlap} = 1/\sqrt{2n}$.

To ensure that no circle lies outside the container, equation (4.21) must be added to the model defined by equations (4.2)-(4.11). As before, we include the polar equivalent of equation (4.21).

$$y_i \geq R \qquad \forall i \in C \tag{4.21}$$

Equation (4.8) is replaced by $0 \leq y_i \leq 1 \ \forall i \in C$ and equation (4.10) is replaced by $0 \leq \theta_i \leq \pi \ \forall i \in P$.

4.2.3.2 FSS heuristic modifications

Equation (4.13), to ensure that the circles are fully inside the container, is replaced by

$$R^* = \min\left\{R, \min\left\{y_i, \{1 - \sqrt{x_i^2 + y_i^2}\} \mid i = 1, ..., n\right\}\right\} \tag{4.22}$$

4.2.4 Circular quadrant container

Here we consider the circular quadrant container as the upper quarter of the unit circle centred at the origin of the Cartesian plane as in Specht [72].

4.2.4.1 Formulation modifications

$R_{overlap}$, the upper bound on the radius, is now defined using $n\pi R^2_{overlap} = \pi 1^2/4$, so $R_{overlap} = 1/\sqrt{4n}$.

For the circular quadrant container we use the mathematical model given in equations (4.2)-(4.11) and we add equation (4.23) to ensure that no circle lies outside the container. As well we include the polar equivalent of equation (4.23).

$$x_i \geq R \qquad y_i \geq R \qquad \forall i \in C \tag{4.23}$$

Equations (4.7) and (4.8) are replaced by $0 \leq x_i \leq 1$ and $0 \leq y_i \leq 1$ $\forall i \in C$ respectively whilst equation (4.10) is replaced by $0 \leq \theta_i \leq \pi/2$ $\forall i \in P$.

4.2.4.2 FSS heuristic modifications

Equation (4.13), to ensure that the circles are fully inside the container, is replaced by

$$R^* = \min\left\{R, \min\left\{x_i, y_i, \{1 - \sqrt{x_i^2 + y_i^2}\} \mid i = 1, ..., n\right\}\right\} \tag{4.24}$$

4.3 Computational results

The computational results for this section are presented according to the container. The unit square container was the single case where we found other work to compare with, the three different sources are: [9, 62, 72]. To the best of our knowledge, there is not yet other work considering the remaining five containers that we have considered, hence we compared our results with the best-known solutions [72] (available in January 2011).

4.3.1 Rectangular containers

Table 4.1 presents results for the case where the container is the unit square. Here the results from our heuristic are compared with those presented by Mladenovic et al [62]. Data shown in Table 4.1 displays a set of 19 instances where n the number of circles follows a sequence in steps of five, starting with 10 circles and finishing with 100 circles. The Best-known† solution is as considered in [62] and refers to the inverse of the best-known solution at that time and here is given exactly as in [62]. The accuracy measure is the percentage deviation, finally we have the computational time.

This table clearly indicates that for this case our FSS heuristic produces results of much better quality having a percentage deviation -0.14 than any of the three approaches (RD, M_C, M_P) of [62]. Regarding computation times, let us recall from Section 3.5.2 that times from the three approaches (RD, M_C, M_P) are presented as in [62],

which are the average time per replication (over 50 replications) from different initial solutions, whilst we present the total time for 25 replications. Doing a fair comparison we would have to calculate the average of the average total time in seconds spent for our approach by dividing the average total time (479) by 25 which is 19.12 seconds. Although it is the highest computation time from all four approaches it is not far from the highest (15 seconds) from the *RD* approach in [62].

Table 4.1: Comparison with Mladenovic et al [62] for the unit square

n	Best-known†	% deviation from best-known				Total time	Average time as reported in [62]		
		FSS	RD	M_C	M_P	FSS	RD	M_C	M_P
10	6.74757140	0.00	0.00	0.00	0.00	61	0.01	0.01	0.02
15	7.86370315	0.00	0.54	0.54	0.00	50	0.03	0.04	0.05
20	8.97808315	0.00	0.00	1.56	0.00	65	0.05	0.11	0.10
25	10.00000000	0.00	0.00	0.00	0.00	78	0.10	0.24	0.28
30	10.90856809	0.00	0.63	0.63	0.58	99	0.26	0.61	0.57
35	11.86370360	0.00	0.32	0.32	0.59	155	0.38	1.04	1.11
40	12.62837533	0.00	0.09	0.09	0.19	169	1.10	1.94	1.97
45	13.38198309	0.00	0.16	0.16	0.11	237	1.24	2.26	2.54
50	14.01009567	0.00	0.28	1.04	0.28	276	1.87	4.00	3.64
55	14.69391977	0.00	0.61	0.61	0.37	286	3.27	6.15	5.22
60	15.37742112	0.00	0.38	0.38	0.53	350	5.17	8.11	7.13
65	15.82179344	0.00	0.93	0.93	1.09	487	7.50	12.26	10.04
70	16.50255154	0.00	0.36	0.92	0.80	517	13.43	13.12	11.92
75	17.09561268	0.01	0.67	0.73	0.55	621	17.01	18.04	15.37
80	17.43050631	0.00	1.45	1.50	0.77	750	24.95	23.65	23.37
85	17.96028299	0.00	1.39	1.23	1.05	978	33.11	30.44	26.05
90	18.60466847	0.00	0.77	1.25	1.13	1179	43.62	35.85	27.81
95	19.07658639	0.00	0.80	0.94	0.49	1423	51.02	43.49	35.48
100	20.00000000	-2.72	0.00	0.00	0.00	1317	80.80	61.15	47.68
Average		-0.14	0.49	0.68	0.45	479	15.00	13.82	11.60

The second investigation for the unit square container is presented in Table 4.2, here we compare our results with those presented by Birgin and Gentil [9]. The instances considered have $n = 2, 3, ..., 50$ identical circles. Table 4.2 shows the percentage deviation for each approach and the computation time spent in seconds.

In Section 2.1 we mentioned that there are two different points of view to address

the packing problem, here, let us note that the approach used by Birgin and Gentil [9] is that of considering identical unitary circles aiming to minimise the size of the square container, hence they report their solutions in terms of size of the length of the edge of the square container whilst our approach aims to maximise the radii of the small identical circles to fit into the unit square container. This means that if the solution from Birgin and Gentil [9] has edge length $L*$ to fit n unitary circles then in our unit square the circles would have a maximum radius $1/L*$.

From the bottom row of Table 4.2 we can see that although the average percentage deviation presented by the FSS approach is higher, the computational time spent by the FSS approach is considerably lower $1285.84/153.57 = 8.37$ times faster. Birgin and Gentil [9] tests were conducted on a 2.4 GHz Intel Core 2 Quad with 4 GB of RAM memory and running GNU/LINUX operating system, whilst our tests were conducted on a 2.26 GHz Intel Core 2 pc with 4 GB of RAM memory and running in Windows XP operating system. Third and last investigation involving the unit square (US) container is with the best-known solutions [72] (available in January 2011). The comparisons are shown in Table 4.3 which also include the results from the other two rectangular containers considered: a rectangle with length $L = 5$ and width $W = 1$ (R 5×1) and a rectangle with length $L = 10$ and width $W = 1$ (R 10×1). The accuracy measure is the percentage deviation. Data in Table 4.3 shows the best-known solution, the percentage deviation and the total computational time to obtain the results in 25 replications, each replication consisting of 80 iterations. The instances taken into account are comprised of $n = 10, 15, ..., 100, 125, 150, 175, 200, 250$ and 500 circles.

We can see from the bottom row in Table 4.3 that the lowest average percentage deviation (0.01998805) for the instances considered is given by the rectangular container with dimensions length $L = 5$ and width $W = 1$, whilst the highest average percentage deviation (0.05921323) is given by the unit square container. It is worth mention that in Table 4.3 for the case of the rectangle with dimensions 5×1 with 500 circles and for

the rectangle with dimensions 10×1 with 250 and 500 circles the percentage deviation reported is zero as [72] had no results for these cases at the time we carried out the investigation. Regarding about time we can see that the unit square container spent on average the lowest total time with 7295 seconds whilst the rectangular container with dimensions length $L = 10$ and width $W = 1$ spent on average the highest total time with 12968 seconds.

Table 4.2: Comparison with Birgin and Gentil [9]

n	Birgin % dev	FSS % dev	Birgin time	FSS total time
2	-0.0000000001544834	0.0000067205830000	0.00	1598.88
3	-0.0000000000982177	0.0000006071180000	0.01	394.66
4	0.0000000000000000	0.0000000001348033	0.00	30.05
5	0.0000000002184457	0.0000000002216085	0.01	64.67
6	-0.0000000002540556	0.0000067023710000	0.48	67.61
7	-0.0000000000054093	0.0000004992284000	2.29	108.78
8	-0.0000000000318991	0.0000000161926100	4.56	68.19
9	0.0000000002000067	0.0000000003036071	0.69	26.78
10	-0.0000000001543742	0.0000006218509466	43.44	61.22
11	0.0000000001401820	0.0000029846990000	456.84	45.61
12	-0.0000000003057977	0.0000054888090000	126.30	49.97
13	0.0017291990000000	0.0000019844800000	2.96	44.28
14	-0.0000000000262894	0.0000205038000000	13.44	46.06
15	-0.0000000000981741	0.0000007770086094	135.58	50.36
16	0.0000000000000000	0.0000000062423950	0.00	41.08
17	0.0000000000438015	0.0000149937200000	109.89	59.63
18	0.0000000000004565	0.0000086390930000	13.93	57.91
19	0.0000000000033376	0.0000439700700000	5.78	68.92
20	-0.0000000004307032	0.0000002602001092	5.56	65.27
21	-0.0000000000602850	0.0000018143240000	758.56	78.08
22	0.0008923583000000	0.0000011423130000	12.73	76.12
23	0.0000000002100385	0.0000078695700000	4447.51	88.14
24	0.0000000003812286	0.0000019330280000	233.46	86.38
25	0.0000000000000000	0.0000000380855070	0.84	78.39
26	0.0000000001172008	0.0000116351100000	96.25	92.48
27	0.0000000000665820	0.0000009207121000	974.71	104.38
28	0.0000000002294721	0.0000073572230000	80.53	99.91
29	-0.0000000003347006	0.0000101909300000	21.42	102.94
30	0.0000000000126559	0.0000001375722064	28.21	98.53
31	-0.0000000002626017	0.0000037280440000	155.25	118.81
32	0.0000000002334601	0.0000048173860000	438.73	125.09
33	0.0013745760000000	0.0000010999630000	115.67	123.58
34	0.0000000005544745	0.0000699432400000	6428.40	146.50
35	-0.0000000004252372	0.0000026636822117	1802.77	155.47
36	-0.0000000003999967	0.5491094000000000	29.29	162.75
37	0.0005946050000000	0.0000726830400000	6577.27	178.94
38	0.0042973210000000	0.0000003487986000	6889.47	152.03
39	0.0000000000319624	0.0000017520760000	112.60	165.45
40	-0.0000000003571675	0.0000021701146337	2133.86	168.86
41	0.0000000001013102	0.0002256899000000	198.40	174.69
42	-0.0000000002127649	0.0000024032450000	67.83	172.34
43	0.0011217430000000	0.0007067743000000	2055.75	187.31
44	0.0009049288000000	0.3302351000000000	8522.40	198.70
45	0.0000000000197041	0.0000327147457704	510.51	236.80
46	0.0008446683000000	0.0000017148530000	151.07	215.95
47	0.0011539000000000	0.0000511495200000	4731.81	228.19
48	-0.0000000006868933	0.0000357124300000	2214.69	236.64
49	0.0000000005251256	0.0055037630000000	1334.68	245.84
50	0.0170323600000000	0.0000067387512595	12245.42	275.95
Average	0.0006111358814366	0.0180862894302730	1285.84	153.57

Table 4.3: Comparisons with Packomania web site [72] for identical circles inside rectangular container

	Best-known radius			% deviation from best results of FSS			Total time of FSS		
n	U S	R 5×1	R 10×1	U S	R 5×1	R 10×1	U S	R 5×1	R 10×1
10	0.14820432	0.06185032	0.05000000	0.00000062	0.00000031	0.00000000	61	85	40
15	0.12716655	0.05505051	0.03598519	0.00000078	0.00000694	0.00003046	50	63	63
20	0.11138235	0.05000000	0.03109074	0.00000026	0.00000000	0.00000693	65	48	76
25	0.10000000	0.04226996	0.02885380	0.00000004	0.00000972	0.00006879	78	86	140
30	0.09167106	0.03923334	0.02765293	0.00000014	0.00003912	0.00006820	99	108	123
35	0.08429071	0.03761333	0.02693590	0.00000266	0.00000337	0.00002189	155	159	141
40	0.07918675	0.03593823	0.02500000	0.00000217	0.00017827	0.00000001	169	171	122
45	0.07472734	0.03333333	0.02265228	0.00003271	0.00000000	0.00206741	237	176	167
50	0.07137710	0.03117646	0.02133467	0.00000674	0.00004963	0.00033613	276	220	208
55	0.06805536	0.02974024	0.02040995	0.00000207	0.00003851	0.00020284	286	251	268
60	0.06503041	0.02887175	0.01973563	0.00000467	0.00000759	0.00032433	350	309	325
65	0.06320396	0.02820113	0.01924353	0.00002148	0.00001523	0.00016231	487	401	445
70	0.06059669	0.02777780	0.01884091	0.00036655	0.00000436	0.00027852	517	495	538
75	0.05849454	0.02634156	0.01854818	0.00689349	0.00259092	0.00030264	621	608	692
80	0.05737068	0.02513053	0.01834058	0.00000015	0.10504818	0.00000309	750	692	817
85	0.05568018	0.02432442	0.01738610	0.00420337	0.00008201	0.05816296	978	771	1038
90	0.05374995	0.02372096	0.01669075	0.00007667	0.00045144	0.12902497	1179	927	1241
95	0.05242037	0.02323914	0.01615003	0.00224318	0.00028429	0.17491486	1423	1110	1333
100	0.05140107	0.02288111	0.01566314	0.00249622	0.00008262	0.00035285	1317	1259	1640
125	0.04597834	0.02032251	0.01431718	0.08040242	0.13161784	0.00028669	2452	2263	3192
150	0.04214547	0.01896176	0.01324134	0.00400724	0.00000181	0.04666236	4107	3942	6195
175	0.03906110	0.01732794	0.01206593	0.03707206	0.06322381	0.43360267	6374	6717	11735
200	0.03661280	0.01641240	0.01147946	0.29745107	0.00025828	0.11747268	8790	9463	19004
250	0.03287632	0.01463438	0.01011529	0.08660955	0.19570694	0.00000000	18111	20465	56617
500	0.02344549	0.01046709	0.00573487	0.95843434	0.00000000	0.00000000	133443	241386	218047
Average				0.05921323	0.01998805	0.03857414	7295	11687	12968

4.3.2 Other containers

Table 4.4 presents the results for the right-angled isosceles triangle with length $L = 1$, a semicircle with radius one and a circular quadrant with radius one. In this table we can see that some results showing the percentage deviation are negative numbers, this means that the solution given by our heuristic outperformed the best- known solution [72]. For the instances considered in Table 4.4 the right-angled isosceles triangle presented five improvements having on average a percentage deviation of 0.00877214 with an average total time of 6126 seconds, the semicircular container presented the lowest average percentage deviation of -0.00644147 with an average total time of 4540 seconds having eleven instances with better results than the best-known solution, finally the circular quadrant showed five improved instances with an average percentage deviation of 0.05615641 with an average total time of 5900 seconds.

In the light of these improvements we decided to investigate and extend the set of instances with $n = 2, 3, ..., 150$ identical circles for all containers. For computational reasons we did not venture above $n = 150$, other than for the specific values of n shown in Tables 4.3 and 4.4. Obtaining the results seen in Table 4.5 required, in total, approximately 280 h which is nearly 12 days of computation. Although for the new set of instances with $2 \leq n \leq 150$ circles we considered all containers, the unit square container showed no improvements and its results are not in Table 4.5, however it took on average a total time of 1115 seconds. As well the 280 h of computation time is considering the average total time of the unit circle container (637 seconds). Table 4.5 presents a summary for containers that gave improved results, information displayed on Table 4.5 for $2 \leq n \leq 150$ consists of:

- The number of improved cases (an improved case being one where we improved on the best-known results reported in Specht [72]).
- The average % deviation from (previously) best-known of these improved cases.

- The values of n for these improved cases.
- The average % deviation (from previously best-known result) over all the n values considered.
- The average time taken in seconds (averaged over all the n values considered).

We can see from Table 4.5 that the semicircular container with radius one and centred at the origin presented the highest number of improvements (69) for instances whose number of circles ranged between 2 and 150, that is $(69/149) = 0.4631$ which means that the 46.31% of the cases considered were improved. Regarding its average percentage deviation over the improved cases, it is -0.031634857204 whilst its average percentage deviation over all values of $n = 2, 3, ..., 150$ is -0.0071 having the lowest average total time of 695 seconds. Figure 4.2(c) shows the solution associated with Table 4.5 for $n = 19$ circles when the container is the semicircle. All pictures presented in Figure 4.2 are as in figures in Chapter 3 where circles with an "o" in their centre represent circles expressed in Cartesian coordinates whilst circles with a "$*$" in their centre are for circles expressed in polar coordinates.

According to Table 4.5 the circular quadrant container with radius one is the second best case with a high number of improvements (46), that is $(46/149) = 0.3087$ which represents 30.87% of improvement cases. Its average percentage deviation over the improved cases is -0.015183685245 whilst its average percentage deviation over all values of $n = 2, 3, ..., 150$ is of 0.0145. Figure 4.2(d) shows the improved solution associated with Table 4.5 for $n = 29$ circles for the circular-quadrant container.

For the right-angled isosceles triangle container with length $L = 1$ the number of improvements was 18 which represents $(18/149 = 0.1208)$ so 12.08%. For the rectangular container with dimensions $L = 5$ and $W = 1$ we present 9 improvements which is $(9/149 = 0.0604)$ so 6.04% of improved cases. Regarding the rectangular container with dimensions $L = 10$ and $W = 1$ we have 6 improvements $(6/149 = 0.0403)$ so 4.03% of

improved cases. Figures 4.2(b), 4.2(e) and 4.2(f) show the improved solutions for $n = 13$ for the triangular container, $n = 37$ for the rectangular container with dimensions $L = 5$ and $W = 1$, finally for $n = 39$ circles for the rectangular container with dimensions $L = 10$ and $W = 1$. Although the unit square container showed no improvements we present in Figure 4.2(a) a picture with $n = 25$ circles related to result from Table 4.3 where the % deviation when compared with the best-known was of 0.00000004.

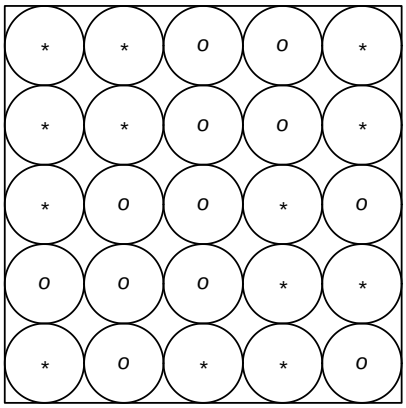
It is clear from Table 4.5 that our FSS heuristic is able to produce results better than previously best-known results. Since Specht [72] is updated with results from a continually running search as well as with results from all authors this is a significant achievement. Some of the results presented here have already been communicated to Specht [72] and can be seen on the Packomania web site.

Table 4.4: Comparisons with Packomania web site [72] for identical circles inside different containers

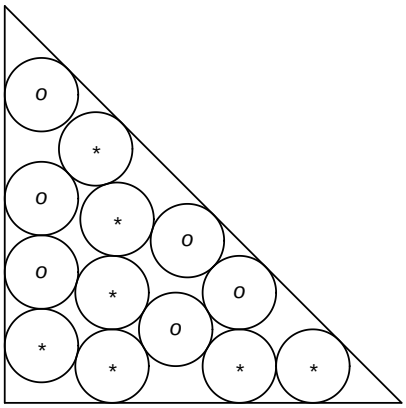
	Best-known radius			% deviation from best results of FSS			Total time of FSS		
n	Triangle	Semicircle	Quadrant	Triangle	Semicircle	Quadrant	Triangle	Semicircle	Quadrant
10	0.10622236	0.18826273	0.13507299	0.00000000	0.00000000	0.00000000	25	26	32
15	0.08761007	0.15862581	0.11021202	0.00000000	0.00000000	0.00000000	35	34	40
20	0.07637899	0.13909585	0.09706486	0.00000000	0.00000000	0.00000000	49	49	50
25	0.06855244	0.12458460	0.08783098	0.00000005	0.00000000	0.00000000	66	61	67
30	0.06306127	0.11396560	0.08056619	-0.00116554	0.00000007	0.00000005	99	82	85
35	0.05870276	0.10578339	0.07455576	0.00000001	0.00000003	0.00000003	127	104	114
40	0.05528479	0.09906682	0.07008873	0.00000165	0.00000000	0.00000001	147	115	128
45	0.05215170	0.09366677	0.06642553	0.00000048	0.00000076	0.00000459	186	139	158
50	0.04978289	0.08909114	0.06309977	0.00000032	-0.00296879	-0.00042110	204	172	193
55	0.04751903	0.08503462	0.06034441	0.00000003	0.00000000	-0.00389984	270	204	235
60	0.04559821	0.08148310	0.05780469	-0.00696771	-0.02792077	0.08300160	329	236	273
65	0.04392293	0.07876654	0.05559034	-0.01848641	-0.00066426	-0.02016617	376	289	328
70	0.04248494	0.07605493	0.05383862	0.00000771	-0.00803026	-0.00001603	422	338	390
75	0.04103285	0.07330218	0.05203938	0.00000271	0.00005501	0.00000001	617	388	463
80	0.03984458	0.07131755	0.05047000	0.00002315	-0.00466258	0.00156236	610	472	560
85	0.03866429	0.06923479	0.04902977	0.02820367	-0.00297505	0.02630948	704	608	633
90	0.03769046	0.06756591	0.04769082	-0.00532499	0.00000005	0.02557222	799	695	721
95	0.03669864	0.06557692	0.04644563	0.01127538	-0.06325532	0.15671220	942	739	826
100	0.03580337	0.06402453	0.04541670	-0.00030914	-0.00458006	-0.00101656	1379	804	935
125	0.03220479	0.05757342	0.04066541	0.02634894	-0.00422906	0.01868178	1875	1462	1757
150	0.02953038	0.05271053	0.03730329	0.01686111	-0.04790578	0.14962182	3132	2374	3112
175	0.02742193	0.04887802	0.03460409	0.13031379	-0.09912597	0.11936323	5202	3891	4870
200	0.02569074	0.04592558	0.03241631	0.03851828	0.01942512	0.02110307	7533	5848	7530
250	0.02305043	0.04112424	0.02913502	0.00000000	0.08580021	0.19391764	14942	11126	14346
500	0.01640334	0.02929358	0.02077970	0.00000000	0.00000000	0.63357992	113086	83242	109647
Average				0.00877214	-0.00644147	0.05615641	6126	4540	5900

Table 4.5: FSS improved results for $2 \leq n \leq 150$

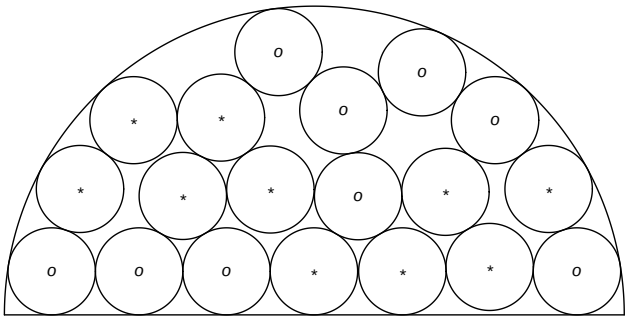
	Rectangle 5×1	Rectangle 10×1	Triangle	Semicircle	Circular quadrant
Number of improved cases	9	6	18	69	46
% of improved cases	-0.026620256247	-0.021473329600	-0.005939179656	-0.031634857204	-0.015183685245
value of n	37 49 74 83 101 131 132 141 148	39 43 76 82 109 139	13 27 30 36 44 48 56 57 60 61 65 68 90 92 100 111 116 130	19 37 41 46 47 49 50 52 53 54 60 64 65 66 68 70 71 74 79 80 81 82 83 85 86 87 88 91 92 93 94 95 97 98 100 104 105 106 107 108 112 113 115 116 118 119 120 121 122 123 124 125 127 130 131 132 133 134 135 136 140 141 142 144 146 147 148 149 150	29 34 37 44 47 49 50 52 55 56 57 58 62 64 65 66 69 70 71 72 73 78 81 84 87 89 91 92 97 99 100 102 103 106 107 108 111 118 119 120 121 129 134 137 139 146
Average % over all different n	0.0263	0.0242	0.0227	-0.0071	0.0145
Average time in seconds	1098	1518	869	695	825



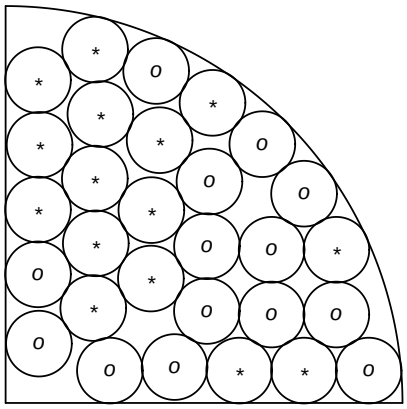
(a) Unit square with 25 circles



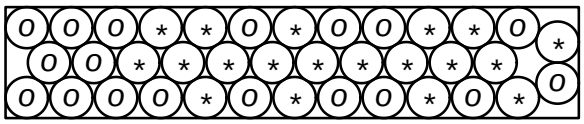
(b) Isosceles triangle with 13 circles



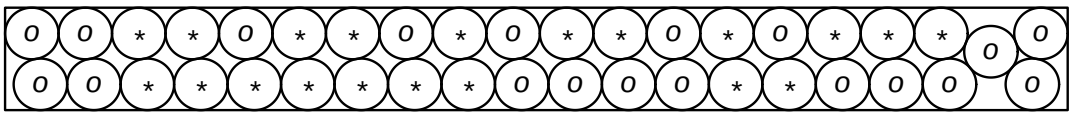
(c) Semicircle with 19 circles



(d) Circular quadrant with 29 circles



(e) Rectangle 5×1 with 37 circles



(f) Rectangle 10×1 with 39 circles

Figure 4.2: Solutions with improvements in different containers

4.4 Conclusions

In this chapter we showed the capabilities of our FSS heuristic in that it can be easily implemented for containers other than the circular one. We presented six different shaped containers: the unit square, a rectangle with $L = 5$ and $W = 1$, a rectangle with $L = 10$ and $W = 1$, a right-angled isosceles triangle with length $L = 1$, a semicircle and a right-circular quadrant with radius 1. The adaptations for each one of the containers were minor and related to the boundaries of the container and the limits on the variables.

Regarding the results presented for instances with $n = 5, 10, ..., 100, 125, 150, 175$, 200, 250 and 500 circles and for the triangle, semicircle and the circular quadrant we observed a number of improvements over the (previous) best-known solutions. Encouraged by those results we decided to increase the number of instances now considering $n = 2, 3, ..., 150$. For computational reasons we did not undertake larger instances as the total time spent was around 12 days of computations. The results for the new instances considered showed improvements over the (previous) best-known; for the rectangular container with dimensions $L = 5$ and $W = 1$ we obtained 9 improvements (6.04% of improvements), for the rectangular container with dimensions $L = 10$ and $W = 1$ we obtained 6 improvements (4.04% of improvements), for the right-angled isosceles triangle we obtained 18 improved cases (12.08% of improvements), for the semicircular container we obtained 69 improvements (46.31% of improvements) and finally for the circular quadrant we obtained 46 improvement (30.87% of improvements).

In general we can say that our FSS heuristic is an effective algorithm when balancing the accuracy of the results with computational time spent. Moreover, it showed the capability of easy adaptation to other shaped containers and gave improvements (now published and available from the web site Packomania [72]) over the (previously) best-known solutions.

Chapter 5

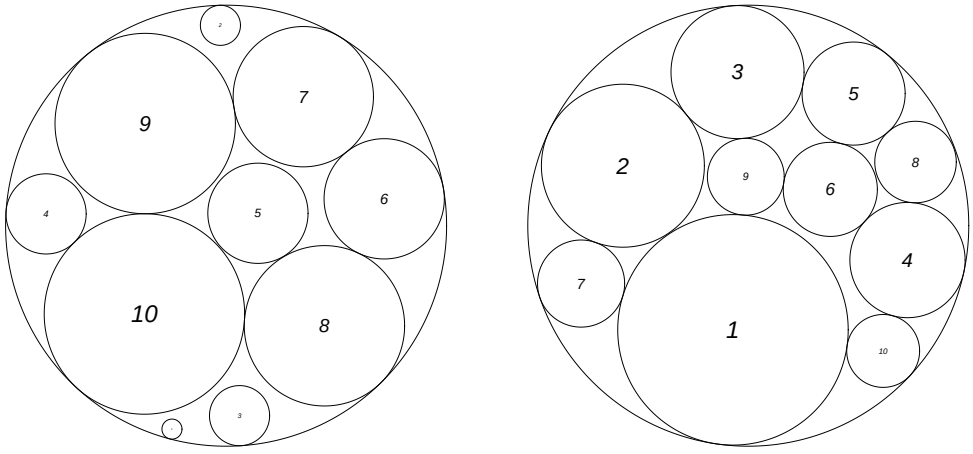
Packing non-identical circles inside a circular container

Addressing the packing problem with non-identical circles leads us to two different approaches: the aim of the first approach is to find the minimum radius of the circular container considering the non-identical circles as being of fixed size, it is characterized by a greedy strategy to resolve the overlaps encountered after the solver solution whilst the second approach is based on a scaled formulation for the unit circle container. The "greedy strategy" approach is given in Appendix A since, computationally, it was less successful than the scaled approach given in this chapter.

This chapter is divided into six sections. In Section 5.2 we present a scaled formulation that is based on reformulation descent considering the unit circle container whilst in Section 5.3 we present the heuristic algorithm developed. In Section 5.4 we present the computational results from the tests considered and in Section 5.6 we give the conclusion to the chapter.

5.1 Non-identical circles

We address two different categories of non-identical circles: large variation instances and small variation instances. We consider an instance a large variation instance if the radii are defined as $R_i = i$ for all $i = 1, ..., n$ and an instance a small variation instance when their radii are defined as $R_i = 1/\sqrt{i}$ for all $i = 1, ..., n$. Generically a ***large variation*** instance means that there is a wide disparity between the size of the circles, a ***small variation*** instance means that there is only a small disparity between the sizes of the circles. In Figures 5.1(a) and 5.1(b) are two examples of non-identical circles inside a unit circle where the difference between large and small variation can be appreciated if we compare the size of the largest circle with the size of the smallest circle in Figure 5.1(a) as compared to Figure 5.1(b). The reason for distinguishing large/small variation instances is that we adopt different algorithmic steps depending upon the type of instance being considered.



(a) Unit circle $R_i = i$, large variation instance

(b) Unit circle $R_i = 1/\sqrt{i}$, small variation instance

Figure 5.1: Circle packing problem with $n = 10$ non-identical circles

5.2 Scaled formulation for the unit circle container

The formulation that we used to address the packing problem with non-identical circles size considers a circular container. This formulation is different from those found in the literature, where the problem is most commonly formulated as considering the radius of each circle as a fixed value, aiming to find the minimum radius of the circular container. Here we regard the circular container as being fixed, namely the unit circle. This means that the size of the container is fixed and even though the radius of each circle has been previously defined as R_i the aim of this approach is to find the ***maximum common scaling factor*** for each circle denoted as ρ. In other words we seek the maximum value of ρ such that non-identical circles of radii ρR_i can be packed inside the unit circle container. As far as we are aware this scaled view of the problem has not been considered before in the literature.

In terms of the unit circle container it is clear that the maximum value of ρ, say ρ_{max}, means that the original circles, of radii R_i, can be packed inside a circular container of minimum radius $1/\rho_{max}$ with the non-identical circles having centre coordinates $(x_i/\rho_{max}, y_i/\rho_{max})$ for $i = 1, ..., n$. This scaled problem and the standard problem in the literature are therefore equivalent. However for some containers, such as rectangles, a triangle, semicircle and circular quadrant, this scaled view of the problem leads to new problems that have not been considered previously in the literature, but these containers will be discussed in Chapter 6. The formulation presented below considers the unit circle container and details about the changes to the formulation and the heuristic for the other containers will be given in detail in the next chapter.

Our maximum common scaling formulation for the circle packing problem with non-identical circles and the unit circle as the container is:

$$\max \quad \rho \tag{5.1}$$

subject to

$$x_i^2 + y_i^2 \leq (1 - \rho R_i)^2 \qquad \forall i \in C \qquad (5.2)$$

$$r_i \leq 1 - \rho R_i \qquad \forall i \in P \qquad (5.3)$$

$$(x_i - x_j)^2 + (y_i - y_j)^2 \geq (\rho(R_i + R_j))^2 \qquad \forall (i,j) \in Q \text{ with } i,j \in C \; i < j \qquad (5.4)$$

$$(x_i - r_j \cos(\theta_j))^2 + (y_i - r_j \sin(\theta_j))^2 \geq (\rho(R_i + R_j))^2 \qquad \forall (i,j) \in Q \text{ with } i \in C \; j \in P \qquad (5.5)$$

$$r_i^2 + r_j^2 - 2 r_i r_j \cos(\theta_i - \theta_j) \geq (\rho(R_i + R_j))^2 \qquad \forall (i,j) \in Q \text{ with } i,j \in P \; i < j \qquad (5.6)$$

$$-1 \leq x_i \leq 1 \qquad \forall i \in C \qquad (5.7)$$

$$-1 \leq y_i \leq 1 \qquad \forall i \in C \qquad (5.8)$$

$$0 \leq r_i \leq 1 \qquad \forall i \in P \qquad (5.9)$$

$$0 \leq \theta_i \leq 2\pi \qquad \forall i \in P \qquad (5.10)$$

$$0 \leq \rho \leq \sqrt{1/\Sigma_{i=1}^{n} R_i^2} \qquad (5.11)$$

The objective function (5.1) maximises the scaling factor associated with each radius R_i. Equations (5.2) and (5.3) are the constraints that ensure that all circle centres [(x_i, y_i) or (r_i, θ_i)] are inside the container, for this particular case, the unit circle. Equations (5.4), (5.5) and (5.6) guarantee that any two circles i and j do not overlap each other. Equations (5.7) and (5.8) represent the limits for the variables that are in the Cartesian system whilst equations (5.9) and (5.10) represent the limits of the variables that are in the Polar system. Finally equation (5.11) represents the limit on ρ (derived from area considerations since we need $\Sigma_{i=1}^{n} \pi(\rho R_i)^2 \leq \pi(1)^2$). The formulation given above is based on the formulation given in López and Beasley [54] for packing identical sized circles in the unit circle and has been considered in this thesis in Chapter 3.

5.3 Heuristic

The heuristic presented here, was first conceived as one based on formulation space search, however after several computational experiments given in detail in Section 5.5 we observed that the scaled formulation gave better results when adopting a reformulation descent approach.

The heuristic developed is divided into two phases: an optimisation phase where we find a local optima and an improvement phase where we improve the solution obtained in the optimisation phase. For the optimisation phase we used the Reformulation descent method whilst for the improvement phase we seek improvement by swapping circles and packing circles in free space.

5.3.1 Optimisation problem

The optimisation problem will be addressed by the reformulation descent (RD) method. We use the mixed Cartesian/Polar formulation as presented in equations (5.1)-(5.11), denoted as $NLP(x^0, y^0, C, P)$ in the pseudocode given below. Even though the result given by the non-linear solver (SNOPT) has a high degree of accuracy we have included a correction step to avoid having non-feasible solutions and the details will be described in Section 5.3.2. Having a mixed formulation we set the cardinality of sets C and P as $|C| = \lfloor n/2 \rfloor$ then $|P| = n \setminus |C|$, after each iteration we will switch the circles in C to P and vice-versa.

5.3.1.1 Subset of circles in the optimisation process

When the radius of the circles are defined by $R_i = i$ (large variation instance) we can see that the smallest circle has radius $R_1 = 1$ whilst the biggest one is n times bigger with radius $R_n = n$. This difference in size is so big that (assuming n is reasonably large) if we exclude circle one with radius $R_1 = 1$ from the optimisation process it

will probably not have an impact on the optimal solution. By this we mean that the optimal packing of circles $i = 2, \ldots, n$ will probably have sufficient free space that circle one can be packed without perturbing the already packed circles. However excluding circle one would decrease the number of non-overlapping constraints in the problem (equations (5.4)-(5.6)), as it would be one less circle to be paired with the other $(n-1)$ circles as part of the set Q. An illustration for the large variation instance case is in Figure 5.1(a) for $n = 10$ non-identical circles.

In the light of this when considering the case where $R_i = i$ we set $\lceil .70n \rceil$ as the number of circles for the optimisation process, that is we will be taking into account just seventy percent of the circles, the biggest circles as in Grosso et al [23]. The packing of the remaining smaller circles (thirty percent of the circles) is done in the insertion process that takes place after the improvement process, the details will be given in Section 5.3.5. Instances of this type are large variation instances.

In contrast, we have the case when radii of the circles are defined by $R_i = 1/\sqrt{i}$ where the difference between radius $R_1 = 1$ and radius $R_n = 1/\sqrt{n}$ is smaller $\sqrt{n}$ times, with $R_1 = 1$ as the largest circle. As an example, if $n = 5$ the radius of circle one is $R_1 = 1$ whilst the radius of circle 5 is $R_5 = 1/\sqrt{5} = 0.4472$, this means that R_1 is $\sqrt{5} = 2.2361$ times larger than R_5. As this difference is not as big as in the previous instance where $R_i = i$, here for $R_i = 1/\sqrt{i}$ we did not include the insertion process and for the optimisation phase we work with the entire set of n circles. Instances of this type are small variation instances. An example of these instances is depicted in Figure 5.1(b).

Essentially for large variation instances we make a computational saving by leaving the "smaller" circles be inserted after the optimisation phase. For small variation instances the nature of the circle sizes means that we consider all of them in the optimisation phase.

5.3.2 Correction step

A correction step is included in our heuristic to ensure that every solution given by the solver is feasible. All solutions must satisfy two conditions: being inside the container and no overlapping circles. If these conditions are not met the appropriate corrections will be applied. At the end of the correction step we should have a feasible solution with an associated value for ρ.

First (working solely in Cartesian coordinates for convenience) we test that for every circle i, with circle centre at coordinates (x_i, y_i), is inside the container, in this case the unit circle. In other words the circle centre must satisfy $x_i^2 + y_i^2 \leq 1$. If for some circle k, we have $x_k^2 + y_k^2 > 1$ then the coordinates (x_k, y_k) will be arbitrarily reallocated inside the container. Computational experience has been that all solutions given by the solver are inside the container, however logically this step is needed to ensure the feasibility of the solution. In addition we define

$$\rho_1 = \min\{(1 - \sqrt{x_i^2 + y_i^2})/R_i | i = 1, ..., n\} \tag{5.12}$$

as the maximum value for variable ρ in equation (5.2) which ensures that all circles are fully inside the unit circle container.

To satisfy the non-overlapping condition, we consider all possible pairs of circles and calculate the distance between their centres keeping the minimum, that is, for any two circles (i, j) with radii R_i and R_j we set

$$\rho_2 = \min\left\{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}/(R_i + R_j) | i < j \ \forall (i, j) \in Q\right\} \tag{5.13}$$

Here ρ_2 represents the maximum value for variable ρ such that no two circles overlap. Finally we take

$$\rho_{current} = \min\{\rho_1, \rho_2\} \tag{5.14}$$

where $\rho_{current}$ is the best value that the variable ρ can take whilst keeping the current solution feasible, as well it represents the maximum scaling factor by which each radius

R_i for all $i = 1, ..., n$ can be multiplied to fit the n circles with known centres $[x_i, y_i]$ inside the unit circle.

5.3.3 The pseudocode

Our pseudocode is shown in Algorithm 5.7 and it consists of two phases. Phase one solves the packing problem through the reformulation descent method using an off-the-shelf optimiser. Sets C and P are defined by randomly placing circle i with $i = 1, ..., n$ either in C or P. At every iteration the initial solution is generated randomly inside the container, in our case the unit circle by choosing r_i uniformly from the interval $[0, 1]$ and θ_i uniformly from interval $[0, 2\pi]$ and expressing their respective values in Cartesian coordinates (x^0, y^0). This initial solution is a starting guess for the non-linear solver SNOPT [18, 35]. Then we have the correction step to guarantee that all constraints are satisfied. The stopping criteria is based on the number of iterations.

Phase two of the algorithm depends on the problem we are solving. If the problem to be solved is a large variation instance then we considered two processes: first the swapping process (considered in Section 5.3.4) followed by the insertion process (detailed in Section 5.3.5). Whereas if the problem to be solved is a small variation instance we only carry out the swapping process in phase two. The reason for this was explained in Section 5.3.1.1.

The swapping process attempts to improve the current best arrangement by swapping circle i with circle j (see Section 5.3.4 for details). If the swap improves the current solution then we keep it and continue the swapping process from the updated solution until there is no further improvement. In Algorithm 5.7 ρ_{max} represent the maximum scaling factor encountered so far, R^{index} is a vector that keeps track of the radius of each circle. Finally in Algorithm 5.7 as part of phase two we have the insertion process, whose function is to determine a location for small circles inside the existing packing of the largest circles, here (x_{best}, y_{best}) represent the coordinates of the small circle packed

during the procedure.

Given this overview of our pseudocode we now consider in detail the swapping and insertion processes.

5.3.4 Swapping process

The swapping process explores a small part of the combinatorial nature of the problem. In Algorithm 5.7 a solution from phase 1 is denoted as (X, Y) with R^{index} being the vector that keeps track of the radii of all circles. For this procedure we will investigate if a small perturbation of a pair of circles would improve current solution.

Assume (without loss of generality) that the n circles have been indexed such that the radii R_i are in order (either ascending order or descending order as convenient, ties broken arbitrarily). The point of assuming this is that we now know that the two circles with radii (size) closest to circle i are circles $i-1$ and $i+1$. Take circles i and $i+1$ with radii R_i and R_{i+1} respectively, and swap them. In other words, we set the centre of circle i to where $i+1$ is centred, and set the centre of circle $i+1$ to where i was centred. Set (X, Y) with $R^{index} = [R_1, R_2, ..., R_{i+1}, R_i, ..., R_n]$ as initial solution for the non-linear solver. The solver will return as solution the coordinates (x, y) and $\rho_{current}$ with $R^{index} = [R_1, R_2, ..., R_{i-1}, R_{i+1}, R_i, R_{i+2}, ..., R_n]$. If $\rho_{current} > \rho_{max}$ then (x, y) with $R^{index} = [R_1, R_2, ..., R_{i-1}, R_{i+1}, R_i, R_{i+2}, ..., R_n]$ is our best solution up to now with $\rho_{max} \leftarrow \rho_{current}$. However if $\rho_{current} \leq \rho_{max}$ we restore R^{index} as $[R_1, R_2, ..., R_{i-1}, R_i, R_{i+1}, R_{i+2}, ..., R_n]$. Whether we have an improved solution or not we continue swapping the circles. Note that we only swap adjacent circles in R^{index}, hence they will always be of similar sizes. This procedure is applied for both large and small variation instances. We terminate the swapping process when there are no two adjacent circles in R^{index} that can be swapped and improve the solution.

Algorithm 5.7 Reformulation descent for the non-identical circle case

Function $(\rho_{max}, X^*, Y^*) \leftarrow RD(n, replication\_limit, iteration\_limit)$

$t\_rep \leftarrow 0$

repeat

$\rho_{max} \leftarrow 0 \;\; R^{index} \leftarrow [R_1, ..., R_n] \;\; |C| \leftarrow \lfloor n/2 \rfloor \; t \leftarrow 0$ {Initialisation}

Phase 1:

repeat

$(x^0, y^0) \leftarrow$ Randomly generate an initial solution (X, Y)

$C \leftarrow \{1, ..., n\} \setminus P; \;\; P \leftarrow n \setminus C$ {set C and P}

$(x, y, \rho) \leftarrow NLP(x^0, y^0, C, P)$

$\rho_{current} \leftarrow Correction(x^0, y^0, x, y, \rho)$ {correct the radius}

$\rho_{max} \leftarrow \max\{\rho_{max}, \rho_{current}\}$ {update ρ_{max}}

$(X^*, Y^*) \leftarrow (x, y)$ {save coordinates associated with ρ_{max}}

$t \leftarrow t + 1$ {update iteration counter}

until $t = iteration\_limit$

$(X^*, Y^*, \rho_{max}) \leftarrow$ Best solution {best solution from phase 1}

Phase 2:

$(X^*, Y^*, \rho_{max}, R^{index}) \leftarrow Swapping(X^*, Y^*, \rho_{max}, R^{index})$ {ρ_{max} best solution after swap}

repeat

$(x_{best}, y_{best}) \leftarrow Insertion(X^*, Y^*, \rho_{max}, R^{index})$ {best coordinates to insert a small circle}

$(X^*, Y^*) \leftarrow (X^*, Y^*) \cup (x_{best}, y_{best})$

until all circles are being packed

$t\_rep \leftarrow t\_rep + 1$ {update replication counter}

until $t\_rep = replication\_limit$

5.3.5 Insertion process by circle tangency

The insertion process by circle tangency procedure has been designed to find a suitable location around the edge of the container where small circles can be inserted when the larger circles have been packed. This procedure is used solely for large variation instances.

During this process we insert one circle at a time in descending order with respect to their radius (so here we assume that the circles have been indexed in ascending radii order so $R_1 \leq R_2 \leq R_3 \leq \cdots \leq R_n$). Let us consider the smaller circles $1, 2, ..., s$ with radii $R_1, R_2, ..., R_s$ respectively, the first circle to enter the insertion process is the largest of these, circle s. To find a location for circle s we look for the space available between two tangent circles (circles that are touching each other) that are also tangent to the container. Circles that are tangential, one to another, are also known as "kissing" circles.

Figure 5.2 illustrates (in grey) where small circles may be located. For every space found we determine the position (coordinates) for circle s to be located in such a fashion that it will always be inside the container but it may be overlapping the two tangent circles, these coordinates are recorded in a list along with their degree of non-overlap. The degree of non-overlap is simply the minimum of the distances between the centres of the two tangent circles and the coordinates found to locate the small circle, it is derived from equation (5.4). From this list we choose the position that satisfies the non-overlapping constraints, (i.e. whose degree of non-overlap is greater or equal to zero). Once we have chosen the position for circle s we continue with circle $s-1$ and so on.

In Figure 5.2 we can see a packing consisting of circles i, j, k and l, the dashed circles in the grey area represent some of the possibilities where we can locate the small circle s. If we take circles k and l we immediately note that circle l is not touching the edge of the container, but as long as the space between circle l and the container is less than the radius of circle s ($1 - \sqrt{x_l^2 + y_l^2} - \rho R_l < \rho R_s$) then we will consider it as a tangent circle.

Determining the position (x, y) to locate the small circle s with known radius R_s

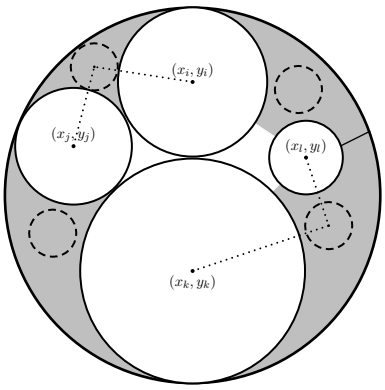


Figure 5.2: Example of the insertion process for the unit circle as container

requires the solution of the non-linear system given by equations (5.15) - (5.17). This system involves the two circles that are tangent to themselves and tangent to the edge of the container, circle i with coordinates (x_i, y_i) and radius R_i and circle j with coordinates (x_j, y_j) and radius R_j.

$$(x - x_i)^2 + (y - y_i)^2 - (\rho_{max} R_s + \rho_{max} R_i)^2 \geq 0 \tag{5.15}$$

$$(x - x_j)^2 + (y - y_j)^2 - (\rho_{max} R_s + \rho_{max} R_j)^2 \geq 0 \tag{5.16}$$

$$x^2 + y^2 - (1 - \rho_{max} R_s)^2 \leq 0 \tag{5.17}$$

These inequalities seek a centre (x, y) for circle s of radius R_s such that circles i and j may touch s and s is inside the container. The solution to this system is given by the non-linear solver. As we examine many possible locations for circle s we will keep the location that best fits, i.e. the one that satisfies the non-overlapping constraints or violates that condition the least. In case our only option is to accept a point $(\hat{x}, \hat{y})$ that does not satisfy the non-overlapping conditions then we call the non-linear solver to get a feasible solution.

5.3.6 A glance into our heuristic

So far we have detailed the steps of our algorithm and given pseudocode for the entire process, now we present a snapshot using $n = 10$ non-identical circles with radii $R_i = i$

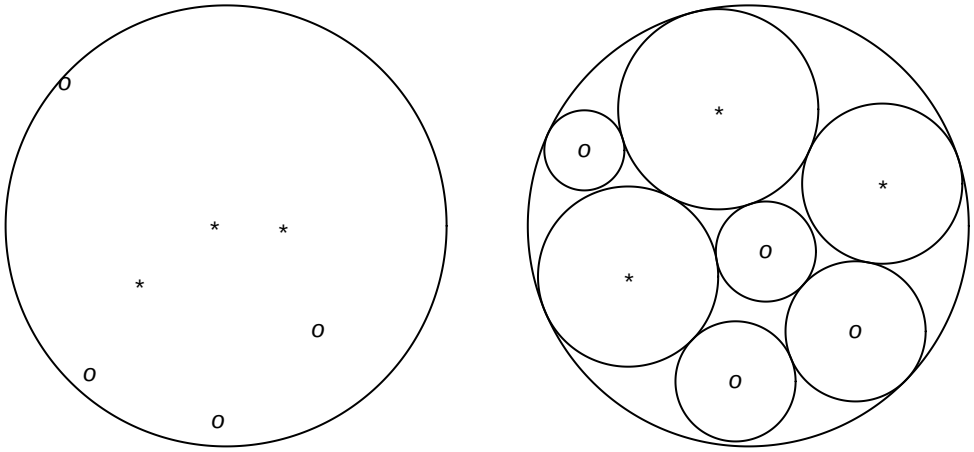
for $i = 1, ..., n$ to be packed inside the unit circle.

Let us recall from Section 5.3.1.1 that for large variation instances for the optimisation process we solely consider the $\lceil 0.70n \rceil$ of the larger circles, so we start (for this particular case) with only 7 circles. In Figure 5.3(a) we present a random initial solution with 7 circles, coordinates with an "o" are for those circles expressed in the Cartesian system whilst those coordinates with an "$*$" are for those circles expressed in the polar system. Figure 5.3(b) represents the solver solution whilst Figure 5.3(c) is the final solution before the swapping process and as well is the solution after the swapping process as in this case there was no difference between the two solutions. In Figure 5.3(d) we present the final solution after the insertion process where we incorporate the three smallest circles which are expressed in Cartesian coordinates and denoted with a "$\cdot$" in their centre.

5.4 Computational results

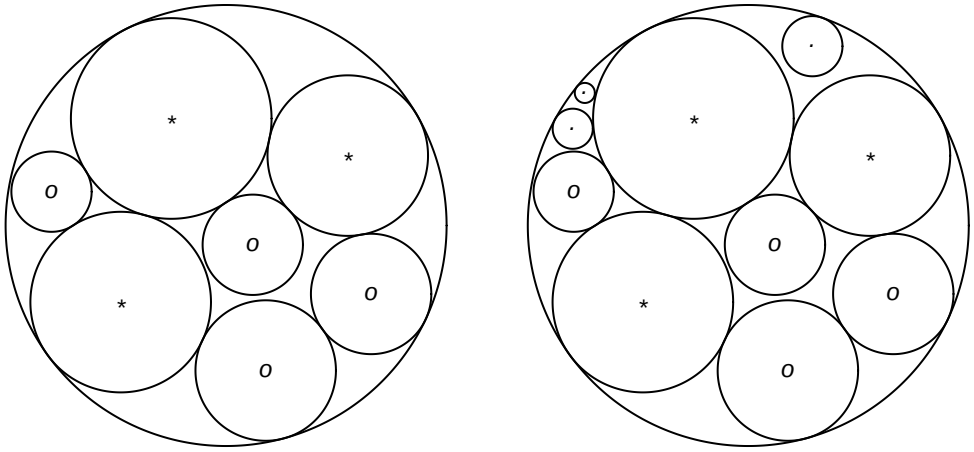
In this section we deal with the circular container. As discussed above we regard this case as being one of finding the maximum common scaling factor such that the non-identical circles can be packed inside the unit circle. Other authors in the literature regard the problem as one of packing fixed sized non-identical circles inside a circular container where the objective is to minimise the radius of the container. Since (also as discussed above) there exists a direct correspondence between these two views of the problem (our solution with maximum common scaling factor ρ_{max} corresponds to a circular container of radius $1/\rho_{max}$) here, for ease of comparison with previous work, we present our results in terms of packing fixed sized non-identical circles inside a circular container where the objective is to minimise the radius of the container. We compare our results against previous results reported in the published scientific literature.

The results presented in this section for our heuristic were produced on an Intel(R) Core(TM) i5-2500 3.30 GHz CPU with 4.00 GB. The algorithm was coded in MatLab 7.9.0 using SNOPT [18, 35] as the non-linear solver. Numerical settings such as the



(a) Random initial solution

(b) Non-linear solver solution



(c) Final solution

(d) Final solution after inserting small circles

Figure 5.3: A snapshot of our heuristic with $n = 10$ non-identical circles with $R_i = i$

number of iterations and the number of replications that the algorithm uses are detailed and justified in Section 5.5, for the moment let us say that we use five replications, each one consisting of 50 iterations.

5.4.1 Comparison with Al-Mudahka et al [5]

In this section we compare our results against those recently reported in Al-Mudahka et al [5] where they proposed a heuristic (TS/NP) based on tabu search and nested partitions of the space search. They ran their tests in a Pentium IV, 2.66 Ghz and 512 Mb of RAM, their heuristic was coded in Fortran that evokes GAMS, which in turn uses CONOPT3 to solve the NLP problems. Results from this comparison are shown in Table 5.1 for the case when the radii of the circles to be packed are defined as $R_i = i$ and in Table 5.2 when $R_i = 1/\sqrt{i}$.

In Table 5.1 column 1 is the number of non-identical circles to be packed inside the circular container, column 2 gives the best-known solution radius reported in Al-Mudahka et al [5], columns 3 and 4 represent the best solution obtained by Al-Mudahka et al [5] and our heuristic, respectively. The measure that we use to compare our results is percentage deviation, which is calculated as $100(R_{best} - R_{BestKnown})/R_{BestKnown}$, where R_{best} represents the best solution found by the approach in question and $R_{BestKnown}$ is the best solution previously known, as given in column 2 and taken from [5]. If the percentage deviation is negative it means that a solution is better than the previous best-known solution.

It is important to note that in Al-Mudahka et al [5] they measured their results considering the deviation $(R_{best} - R_{Best-Known})$ whilst we are using the percentage deviation, therefore to make a fair comparison of the results we present their results as their respective percentage deviation. Column 7 shows computation time as stated in Al-Mudahka et al [5], in column 8 we present the total time spent by our heuristic over all replications for each instance. The last row of Table 5.1 shows the average percentage

deviation and average total time for each approach.

Table 5.1: Results by TS/NP [5] and RD for the case when $R_i = i \ \forall i = 1, ..., n$

n	Best known Radius	TS/NP best solution Radius	RD best solution Radius	TS/NP % deviation	RD % deviation	TS/NP total time (s)	RD total time (min)
5	9.00139774	9.001398	9.00139775	0.00000	0.00000011	426	7.68
6	11.05704039	11.05704	11.05704040	0.00000	0.00000009	686	11.12
7	13.46211067	13.46211	13.46211068	0.00000	0.00000007	1511	12.02
8	16.22174667	16.22175	16.22174668	0.00000	0.00000006	2551	17.14
9	19.23319390	19.39734	19.23319391	0.85347	0.00000005	4051	23.4
10	22.00019301	22.34516	22.00019301	1.56803	0.00000000	5760	23.97
11	24.96063428	24.96063	24.96063429	0.00000	0.00000004	2094	30.34
12	28.37138943	28.67863	28.37138944	1.08292	0.00000004	3548	38.32
13	31.54586701	32.00719	31.54586702	1.46238	0.00000003	4054	48.42
14	35.09564714	35.41261	35.09564714	0.90313	0.00000000	6146	49.5
15	38.83799550	39.00243	38.83799682	0.42337	0.00000340	6696	65.06
16	42.45811643	42.92185	42.45811644	1.09221	0.00000002	9684	81.95
17	46.29134211	46.77237	46.34518193	1.03914	0.11630646	10168	87.73
18	50.11976262	50.65635	50.20889346	1.07062	0.17783572	14312	103.73
19	54.24029359	55.02744	54.36009421	1.45123	0.22087015	14925	122.49
20	58.40056747	59.04547	58.48047359	1.10427	0.13682422	19825	125.29
21	62.55887709	63.49768	63.00078332	1.50067	0.70638453	5923	148.74
22	66.76028624	68.10291	66.96471591	2.01111	0.30621449	6636	179.37
23	71.19946160	72.70501	71.69822657	2.11455	0.70051790	7209	216.97
24	75.75270412	76.49105	76.12311970	0.97468	0.48898001	8552	216.74
25	80.28586443	81.56595	80.81682360	1.59442	0.66133581	11409	259.49
26	85.07640122	86.43809	85.48743800	1.60055	0.48313842	12062	305.33
27	89.79218156	91.15366	90.93173506	1.51626	1.26910103	13657	309.07
28	94.54998647	96.34813	95.64064140	1.90179	1.15352204	14364	365.51
29	99.51231790	101.7251	100.72003130	2.22364	1.21363207	15185	424.25
30	104.57855508	107.1161	105.88817223	2.42640	1.25228078	20745	427.63
31	109.77194698	111.8996	111.07712597	1.93829	1.18899138	21424	499.55
32	114.86543833	117.6701	116.61226677	2.44173	1.52076070	22781	574.08
Average				1.22481	0.41416784	9514	170.53

One point to note with respect to Table 5.1 is that different authors in the literature give their results to a different number of decimal places. This needs very careful consideration when comparing results (and when calculating percentage deviations). For example in Table 5.1 consider $n = 7$ for which the best-known radius is 13.46211067. At first sight the result of 13.46211 from TS/NP appears better, since it is a lower radius. However this apparent difference only arises because in their work [5] dealing

with TS/NP the authors have given results only to five decimal places. Rounding the best-known result to five decimal places makes it clear that the two radii are identical, hence the percentage deviation of zero seen for this instance for TS/NP in Table 5.1. Comparing the $n = 7$ result for TS/NP with that for RD (so comparing 13.46211 against 13.46211068) raises the same issue, since our RD result appears worse, but in fact when rounded to the same number of decimal places as the result given by TS/NP it is the same. The percentage deviations given for TS/NP in Table 5.1 have been calculated using appropriate rounding.

RD outperformed TS/NP in that for all instances from $n = 5$ to $n = 32$ in Table 5.1 (after consideration of rounding, so performing a fair comparison by utilising the same number of decimal places) the radius from our RD heuristic is better (or equal) to radius given by the TS/NP approach. In other words, with respect to quality of solution, our heuristic dominates (for all the problems examined) the heuristic of Al-Mudahka et al [5]. Over all instances the average percentage deviation for the TS/NP approach of [5] is 1.22481 whilst for RD it is 0.41416784. With respect to computation time the average for [5] in Table 5.1 is 9514 seconds = 159 minutes, similar to the average for RD (albeit on different pcs). No results for $n \geq 32$ were reported in [5]. Nevertheless we considered as our largest instance $n = 35$ circles, the best result from our RD heuristic is a radius of 132.76520032 obtained in a total of 766.65 minutes, a picture showing the result obtained can be seen in Figure 5.4(a).

Table 5.2 has eight columns as Table 5.1, here we are considering that the radii of the circles to be packed are defined by $R_i = 1/\sqrt{i}$. The percentage deviations given in this table have been computed after consideration of rounding (necessary here since the best-known result is only given in [5] to four decimal places, so the results from TS/NP and RD have been rounded to four decimal places before the computation of the percentage deviation). For the 14 instances considered in [5] the results obtained by our RD heuristic dominate the results of [5] in that for all instances we (to the six decimal

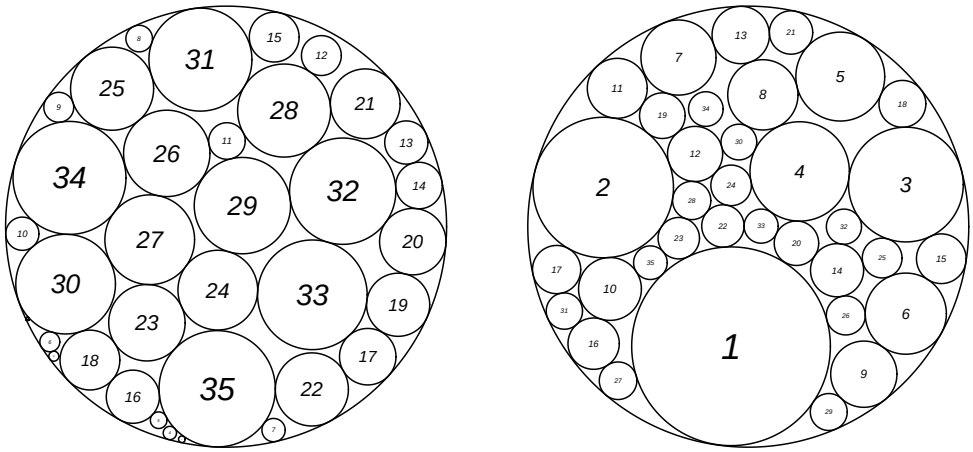
places of [5] shown in Table 5.2) obtain an equal (or better) solution than [5]. Overall the average percentage deviation in the last row of Table 5.2 for the TS/NP approach is -0.3777 whilst for our RD heuristic it is -0.6294. Computation times are again similar (an average of 24316 seconds, so 405.27 minutes, as compared to 331.82 minutes; albeit on different pcs). A picture showing the result obtained by our RD heuristic for $n = 35$ can be seen in Figure 5.4(b).

Table 5.2: Results by TS/NP [5] and RD for the case when $R_i = 1/\sqrt{i}\ \forall i = 1, ..., n$

n	Best known Radius	TS/NP best solution Radius	RD best solution Radius	TS/NP % deviation	RD % deviation	TS/NP total time (s)	RD total time (min)
5	1.7516	1.751552	1.75155246	0.0000	0.0000	1270	11.19
6	1.8101	1.810077	1.81007694	0.0000	0.0000	3169	14.7
7	1.8387	1.838724	1.83872407	0.0000	0.0000	5152	18.64
8	1.8613	1.864532	1.85840095	0.1719	-0.1558	8428	23.3
9	1.8900	1.878813	1.87881276	-0.5926	-0.5926	10489	28.92
10	1.9244	1.920960	1.91343552	-0.1767	-0.5716	14739	35.63
12	1.9696	1.959311	1.95197444	-0.5229	-0.8936	10919	55.87
14	2.0173	2.006005	1.98606765	-0.5602	-1.5466	14885	83.03
16	2.0464	2.031021	2.02213023	-0.7525	-1.1875	22418	114.52
18	2.0664	2.065764	2.05613259	-0.0290	-0.4985	25499	162.47
20	2.1050	2.083187	2.08237082	-1.0356	-1.0736	38164	235.95
25	2.1642	2.143192	2.13929005	-0.9703	-1.1505	30929	595.67
30	2.2008	2.188324	2.18190380	-0.5680	-0.8588	63970	1173.55
35	2.2259	2.220271	2.21958858	-0.2516	-0.2830	90387	2092.05
Average				-0.3777	-0.6294	24316	331.82

5.4.2 Comparison with other work

In Table 5.3 we compare the results obtained by our RD heuristic with those presented by Grosso et al [23], Liu et al [51], Lu and Huang [56] and Zhang and Deng [80]. Those authors considered 13 instances of non-identical circles whose radii can be found in [23, 51, 56, 80] inside a circular container. Table 5.3 consists of seven columns; column one records the instance number, column two indicates how many unequal circles each



(a) Unit circle $R_i = i$ (b) Unit circle $R_i = 1/\sqrt{i}$

Figure 5.4: Circle packing problem with $n = 35$ non-identical circles

instance has, column three indicates where previous results for the instance can be found, column four indicates the best solution (radius) from those presented in [23, 51, 56, 80], column five indicates the best radius obtained by our RD approach, in column six we report the percentage deviation whilst in column seven we report the total time spent by our heuristic. Here, because of the varying number of decimal places quoted in the best-known solution, we have in each instance rounded the RD solution to the same number of decimal places before computing the percentage deviation. With regard to quality of result note that we are comparing our RD heuristic against the best result obtained from four different heuristics [23, 51, 56, 80]. We can see in the last row of Table 5.3 that the average RD percentage deviation is -0.05 that is, overall our RD approach produces better results than previous work. With respect to computation time an average time of just over one hour is not excessive in our view.

Figure 5.5 illustrates two out of four instances (instances 1 and 2 in Table 5.3) for which we improved over previously best-known result: in Figure 5.5(a) we have 10 non-identical circles whose radii are $R_1 = R_2 = 10$, $R_3 = 11$, $R_4 = R_5 = R_6 = 20$, $R_7 = 31.5$, $R_8 = 32$, $R_9 = 40$, $R_{10} = 50$, in Figure 5.5(b) we present the case for 11 non-identical

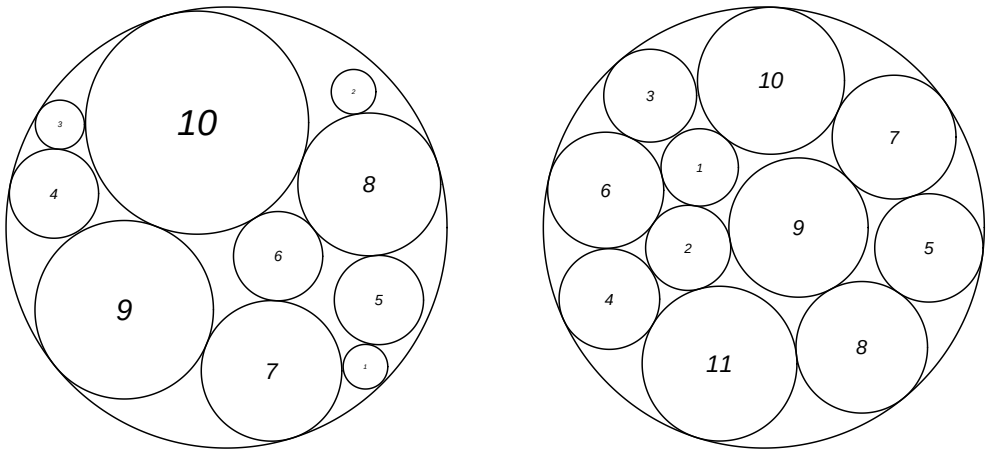
circles whose radii are defined as $R_1 = 21$, $R_{i+1} = R_i + 1$, $i = 1, ..., 10$, lastly in Figure 5.5 we have 16 non-identical circles whose radii are $R_1 = 21$, $R_{i+1} = R_i + 1$, $i = 1, ..., 15$.

Table 5.3: Comparing RD with other work in the literature for the case of non-identical circles inside a circular container

Instance	n	Reference	Best-known solution	RD solution	% deviation	Total time (min)
1	9	[56]	24.143	24.14213562	-0.004	7.94
2	10	[56]	98.90	98.83543170	-0.06	26.14
3	10	[51, 56]	99.8850	99.88507689	0.0000	26.47
4	11	[56]	57.17	56.98440037	-0.33	67.60
5	11	[23, 51]	60.7099	60.70996281	0.0002	59.87
6	12	[56, 80]	215.47	215.47005384	0.00	7.43
7	14	[23, 51]	113.5587	114.57807922	0.8977	139.53
8	15	[23, 51, 56, 80]	38.8379	38.83799682	0.0003	65.06
9	16	[56]	130.00	128.44302642	-1.20	205.86
10	17	[23, 51, 56, 80]	49.1873	49.22967556	0.0862	168.78
11	17	[51, 56, 80]	241.4214	241.42135624	0.0000	9.59
12	25	[51]	21.5470	21.54700538	0.0000	34.56
13	28	[51]	21.5470	21.54700538	0.0000	6.10
Average					-0.05	63.46

5.4.3 Comparing with Packomania website [72]

We have mentioned previously that Packomania [72] is a website dedicated to the packing problem. It considered several different shaped containers for identical and non-identical circles. This website is frequently updating the best-known results, these come from different researchers investigating different approaches to obtain better results than those reported (as well as a continually search). In Table 5.4 we present the comparison between the best results from our heuristic algorithm and the best-known results reported in Packomania [72] (February 2012) for the case of non-identical circles for the large variation instances ($R_i = i$). We considered instances with $n = 5$ up to $n = 35$ circles



(a) 10 non-identical circles (b) 11 non-identical circles

Figure 5.5: Circle packing problem for different instances

inside the unit circle container, we use the % deviation as a measurement of accuracy with respect to the best-known results and we report the total time spent in each instance.

Results from Table 5.4 indicate that on average the accuracy of a solution with $n = 5$ to $n = 35$ circles obtained with our heuristic algorithm is 0.42951263 in 191.09 minutes which is around 3 hours 11 minutes. Although for instances with $n = 5$ to $n = 16$ circles the % deviation is zero, from instances with $n = 17$ to $n = 35$ the % deviation increases ranging from 0.11630644 to 1.55960440.

Regarding the small variation instances we present a comparison in Table 5.5 reporting the number of circles in each instance, the best-known results (Packomania [72]), the best results from our RD approach, the % deviation and the total computational time spent. As in the previous table, here we see that for instances with $n = 5$ to $n = 10$ circles the % deviation is zero whilst for instances with $n = 12$ to $n = 35$ circles the % deviation ranges from 0.11031804 to 1.62248045. Results from the bottom row in Table 5.5 indicate that on average a solution from our heuristic has 0.59493348 accuracy and may be obtained in 331.82 minutes, which is around 5 hours 31 minutes. It is worth mentioning again that the best-known solutions are the results from many

Table 5.4: Comparing RD with Packomania website [72] for large variation instances ($R_i = i \ \forall i = 1, ..., n$)

n	Best known Radius	RD best solution Radius	RD % deviation	RD total time (min)
5	9.00139775	9.00139775	0.00000000	7.68
6	11.05704040	11.05704040	0.00000000	11.12
7	13.46211068	13.46211068	0.00000000	12.02
8	16.22174668	16.22174668	0.00000000	17.14
9	19.23319391	19.23319391	0.00000000	23.40
10	22.00019301	22.00019301	0.00000000	23.97
11	24.96063429	24.96063429	0.00000000	30.34
12	28.37138944	28.37138944	0.00000000	38.32
13	31.54586702	31.54586702	0.00000000	48.42
14	35.09564714	35.09564714	0.00000000	49.50
15	38.83799551	38.83799682	0.00000339	65.06
16	42.45811644	42.45811644	0.00000000	81.95
17	46.29134212	46.34518193	0.11630644	87.73
18	50.11976262	50.20889346	0.17783572	103.73
19	54.24029359	54.36009421	0.22087015	122.49
20	58.40056748	58.48047359	0.13682420	125.29
21	62.55887709	63.00078332	0.70638452	148.74
22	66.76028624	66.96471591	0.30621449	179.37
23	71.19946161	71.69822657	0.70051789	216.97
24	75.74914258	76.12311970	0.49370475	216.74
25	80.28586444	80.81682360	0.66133580	259.49
26	84.97819107	85.48743800	0.59926780	305.33
27	89.75096268	90.93173506	1.31560971	309.07
28	94.52587710	95.64064140	1.17932183	365.51
29	99.48311156	100.72003130	1.24334645	424.25
30	104.54116445	105.88817223	1.28849510	427.63
31	109.62924066	111.07712597	1.32071088	499.55
32	114.82150552	116.61226677	1.55960440	574.08
35	130.85309617	132.76520032	1.46126015	766.65
Average			0.42951263	191.09

different sources and different approaches hence they represent the gold standard.

5.5 Algorithmic variations

This section is concerned with the exploration of algorithmic variations for the two different instances that we are addressing: large variation instances and small variation

Table 5.5: Comparing RD with Packomania web site [72] for small variation instances ($R_i = 1/\sqrt{i}\ \forall i = 1, ..., n$)

n	Best known Radius	RD best solution Radius	RD % deviation	RD total time (min)
5	1.751552455	1.751552455	0.00000000	11.19
6	1.810076939	1.810076939	0.00000000	14.70
7	1.838724068	1.838724068	0.00000000	18.64
8	1.858400955	1.858400955	0.00000000	23.30
9	1.878812755	1.878812756	0.00000002	28.92
10	1.913435515	1.913435515	0.00000000	35.63
12	1.949823437	1.951974444	0.11031804	55.87
14	1.981496158	1.986067647	0.23070898	83.03
16	2.007108056	2.022130232	0.74844878	114.52
18	2.031854924	2.056132595	1.19485259	162.47
20	2.051442268	2.082370823	1.50764927	235.95
25	2.106013266	2.139290050	1.58008426	595.67
30	2.153169187	2.181903797	1.33452633	1173.55
35	2.184151157	2.219588583	1.62248045	2092.05
Average			0.59493348	331.82

instances. Several tests were carried out in order to determine if it was appropriate to use the Q set, the number of iterations per replication, the number of replications and we as well present results from considering single formulations.

5.5.1 Q set

One of the algorithmic variations concerns the Q set, below we explain the reason for not adopting a strategy to reduce its size as we did in Chapter 3.

A fundamental difference between the current heuristic in this chapter and that presented in Chapter 3 is the size of the Q set, whilst in Chapter 3 we proposed a strategy to reduce the size of the Q set (which was satisfactory for the case of identical circles), that strategy has proven to be less effective when the circles to be packed have different radii. Recalling from Section 3.3.2, in order to decide which pairs of circles will be elements of the Q set we used a Δ factor. The main role of Δ is to generate a range of movement around the centre of every circle coordinates in the current solution

to determine a subset of non-overlapping constraints over all possible non-overlapping constraints. In other words, the idea is to prevent the overlaps to happen, thus we only consider the subset of non-overlapping constraints that given the current locations of the centres of the circles, are more likely to overlap.

In this case, for any instance considered all circles will have different size, then to allow them to move far away enough from their current centre coordinate we need to use a Δ factor which depends on the number of circles to be packed n and a numerical factor denoted as δ. To illustrate let us consider a case with $n = 5$ circles, $\delta = 0.05$ then $\Delta = \delta_n = 0.25$. Hence all five circles, although every one has a different size from the other, their centre coordinates are allocated a range of movement of 0.25 in every direction. In order to determine the appropriate numerical value δ for Δ for the case of large variation instances we conducted a series of tests in which δ arbitrarily ranged from 0.05 to 0.09. The tests consisted of running our heuristic with $n = 5, 10, 15$ and 20 non-identical circles, in Table 5.6 we present the results in terms of the average % deviation.

We can see from Table 5.6 for the case of large variation instances that all attempts for which Δ factor depends on δ and the number of circles n give high average % deviation, however for independent values of Δ such as the one given at the bottom row of Table 5.6 where Δ is merely a constant 2 gives the lowest average % deviation of all tests considered. This has a deeper meaning, considering $\Delta = 2$ and the unit circle container implies that no matter where each circle is positioned, all possible pairs of circles may overlap and so all possible pairs of circles are elements of Q set.

However, for the small variation instances most of the % deviation are around 0.8 whether Δ depends on the number of circles or not, although with an independent value for Δ such as 2 we obtained the lowest % deviation. Hence for the case of small variation instances we as well deal with all $n(n-1)/2$ pairs of circles as elements of the Q set. From now on all tests conducted considered all possible pairs of circles as elements of

the Q set.

Table 5.6: Tests to determine Δ factor

Δ variations	large variation instances % deviation	small variation instances % deviation
$.05n$	1.071668960	15.239515072
$.06n$	0.857046726	0.895637569
$.07n$	0.810419474	0.816206424
$.08n$	1.402197273	0.882275491
$.09n$	1.799760639	0.851911411
2	0.314269354	0.809402588

5.5.2 Number of iterations

In order to determine the number of iterations for our heuristic for the large variation instances we conducted a test considering a set of instances with $n = 5, 10, 15$ and 20 non-identical circles. We ran our heuristic for each instance for four different iteration limits: $25, 50, 75$ and 100. We used the % deviation as indicator of the quality of the results.

Searching for insight about the behaviour of our heuristic with respect to the number of iterations we have taken the minimum % deviation up to every iteration for all four instances ($n = 5, 10, 15$ and 20), with these four minimum % deviations we calculated the average and the results are presented in Figure 5.6.

To clarify let us consider four moments depicted in Figure 5.6(a), we see that in iteration one the average of the minimum of all % deviation for $n = 5, 10, 15$ and 20 non-identical circles is close to 1, in fact is 0.9615, after 25 it has drop to 0.4134 and from iteration 50 to 100 the average % deviation is 0.3424.

In Table 5.7 we present in column one the number of iterations being considered and denoted by k, in column two we present the minimum average % deviation given by the set of instances with $n = 5, 10, 15$ and 20 circles after k iterations, column three shows

the average % deviation up to the respective iteration limit, finally in column four we give the total time in minutes spent in the test.

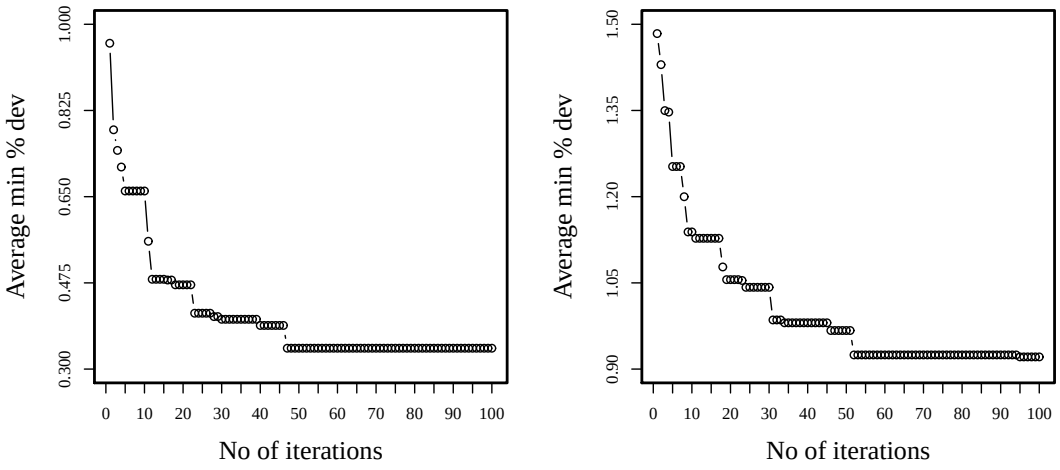
Table 5.7: Tests to determine number of iterations

	Large variation instances			Small variation instances		
number of iterations (k)	Min average % deviation	Average % deviation	total time in min	Min average % deviation	Average % deviation	total time in min
25	0.413391521	0.568639944	26.65	1.042292996	1.167074891	6.52
50	0.342351769	0.479125366	53.30	0.967103469	1.078931979	20.81
75	0.342351769	0.433534167	79.94	0.924653209	1.028071726	56.38
100	0.342351769	0.410738568	106.59	0.921191368	1.002009386	139.43

Results from Table 5.7 indicate that the minimum average % deviation 0.342351769 is obtained between iteration 25 and iteration 50, it stays at the same value until iteration 100, that is there were no further improvements after iteration 50. However, the average % deviation from Table 5.7 indicate that on average with 100 iterations it is more likely that our heuristic algorithm obtain better quality results: having an average % deviation 0.568639944 in 26.62 minutes for 25 iterations and 0.410738568 in 106.59 minutes for 100 iterations, this behaviour can be seen in Figure 5.6(a). As always, balancing quality of results against computation time is a matter of judgement but here 50 iterations seems appropriate.

We carried out the same test for the small variation instances, the results are presented in Table 5.7 indicating the minimum average % deviation per iteration, the average % deviation up to the respective iteration limit and the total computational time spent. These results indicate that on average with 50 iterations we may obtain 1.078931979 % deviation in 20.81 minutes, whilst with 100 iterations the % deviation is 1.002009386 in 139.43 minutes. Here we can see that there is not a significant difference between the % deviation obtained after 50 iterations and that obtained after 100 iterations. However adopting 100 iteration in our algorithm would take $139.43/20.81 = 6.7$ more time. In Figure 5.6(b) we show the behaviour of our algorithm in terms of the average % devi-

ation per iteration, we can see that is from iteration 50 that we obtained the lowest % deviation. Hence for the small variation instances we set the number of iterations to 50..



(a) Large variation instances, 100 iterations (b) Small variation instances, 100 iterations

Figure 5.6: Average % deviation per iteration for large variation instances of $5, 10, 15, 20$ circles

5.5.3 Number of replications

Regarding the number of replications we conducted a test considering the four instances as before ($n = 5, 10, 15$ and 20 circles), running our algorithm with the Q set with all possible pairs of circles, 50 iterations with $5, 10$ and 15 replication limits.

Table 5.8 shows the results of the test, we present the minimum % deviation after k replications, the average % deviation up to the respective replication limit and the total time given in minutes. Data from columns two to four correspond to large variation instances whilst data from columns from five to seven are for small variation instances.

Results from Table 5.8 indicate that for large variation instances the minimum % deviation 0.045310242 correspond to 15 replications with 803.81 minutes total time, moreover it shows based on the average of the minimum % deviation per replication that with 15 replications we obtained a lowest average minimum % deviation 0.135359650. Despite

the fact that the average % deviation for 15 replications is 0.207935793/0.135359650 = 1.536173 times lower than the average % deviation with 5 replication, the total time spent with 15 replications takes 803.81/270.07 = 2.976302 times than with 5 replications. With

Table 5.8: Tests to determine the number of replications

	Large variation instances			Small variation instances		
number of replications	Minimum % deviation	Average % deviation	total time in min	Minimum % deviation	Average % deviation	total time in min
5	0.171330575	0.207935793	270.07	0.593828693	0.631042576	563.53
10	0.071902304	0.169747530	536.50	0.572037683	0.608437022	1121.55
15	0.045310242	0.135359650	803.81	0.475783808	0.571863642	1681.18

respect to small variation instances, we can see that with 15 replications we obtain a minimum % deviation 0.475783808, on average the % deviation is 0.571863642 in 1681.18 minutes, whilst with 5 replications we obtain a minimum % deviation 0.593828693, on average the % deviation is 0.631042576 in 563.53 minutes. This means that considering 15 replications for small variation instances we may obtain results that on average might be 0.631042576/0.571863642 = 1.103484 times better however, in terms of total time 15 replications takes 1681.18/563.53 = 2.983302 more times than 5 replications.

In conclusion based on the insight from the previous test and balancing accuracy with total computational time we adopted 5 as the number of replications for our heuristic for both large and small variation instances.

5.5.4 Single formulations

We considered single formulations in order to determine the effectiveness of the mixed formulation used for the packing problem of identical circles for the case of non-identical circles. The test conducted consisted of running our heuristic algorithm setting 5 replications with 50 iterations each for the case of large variation instances with $n = 5, 10, 15, 20$ non-identical circles using individually a Cartesian and a polar formulation respectively. Table 5.9 gives the results from the test: in columns two, four and six we present the %

deviation with respect to each formulation considered: Cartesian only, polar only and a mixed formulation with Cartesian and polar coordinate systems (as already used earlier in this chapter). In columns three, five and seven we give the total computational time in minutes that each approach took.

Table 5.9: Testing single formulations with a mixed formulation for large variation instances $R_i = i$

n	Cartesian only % deviation	Total time (min)	Polar only % deviation	Total time (min)	Cartesian & Polar % deviation	Total time (min)
5	0.000000000	45.65	0.000000000	13.41	0.000000000	7.68
10	0.000000001	238.42	0.000000000	37.52	0.000000000	23.97
15	0.208280486	589.18	0.197818272	95.91	0.000003388	65.06
20	0.857517317	1204.30	0.339501056	181.18	0.136824198	125.29
Average	0.266449451	519.39	0.134329832	82.01	0.034206896	55.50

The results obtained from all three approaches indicate that as the mixed formulation gives the lowest % deviation it is the best option to use in order to obtain a better solution. Regarding single formulations we can see that for the two aspects that we are considering: quality of solution and total time spent, the polar formulation dominates the Cartesian formulation. The % deviation for the polar formulation is 0.134329832 whilst the % deviation for Cartesian formulation is almost twice that, 0.266449451. In terms of average total time, polar formulation gives a solution in 82.01 minutes, whilst Cartesian formulation takes on average a total time of 519.39 minutes, that is approximately 8.65 hours. From this test we can infer that not only the mixed formulation used for the scaled formulation for the non-identical circles gives better quality solutions but it decreases the computational time need to obtain a solution.

Conducting the same test for the small variation instances we present the results in Table 5.10. Column one shows the number of circles to be packed, columns two, four and six show the minimum % deviation for every case with $n = 5, 10, 15$ and 20

circles for the Cartesian single formulation, the polar single formulation and the mixed formulation respectively. Columns three, five and seven give the total time spent to obtain the results. In the last row of Table 5.10 we present the average % deviation and the average total time for every approach.

Table 5.10: Testing single formulations with a mixed formulation for small variation instances $R_i = 1/\sqrt{i}$

n	Cartesian only % deviation	Total time (min)	Polar only % deviation	Total time (min)	Cartesian & Polar % deviation	Total time (min)
5	0.000000000	41.36	0.000000000	45.43	0.000000000	11.19
10	0.048676620	153.90	0.000292613	70.05	0.000000001	35.63
15	0.817965905	430.89	1.039114235	119.41	0.704579483	137.98
20	1.370449971	1032.13	1.564140072	307.47	1.507649269	235.95
Average	0.559273124	414.57	0.650886730	135.59	0.553057188	105.19

Regarding the average % deviation from the three approaches from Table 5.10 the accuracy of the results seems to be similar: 0.559273124 for the Cartesian formulation, 0.650886730 for the polar formulation whilst 0.553057188 % deviation for the mixed formulation. However, on average the Cartesian formulation takes 414.57 minutes, that is approximately 6.9 hours, by contrast the polar formulation takes on average 135.59 minutes, which is around 2.2 hours, qhilst for our mixed formulation the average time spent is 105.19 minutes which is around one hour 45 minutes. These results indicate that although the quality of a solution may be approximately the same for any formulation considered, the fastest approach is given by the mixed formulation.

5.6 Conclusions

In this chapter we presented a new approach "the scaling formulation model " to solve the circle packing problem with non-identical circles inside the unit circle. It was used to solve two different categories of instances: the large variation instances ($R_i = i$ with

$i = 1, ..., n$) and the small variation instances ($R_i = 1/\sqrt{i}$ with $i = 1, ..., n$). The proposed heuristic algorithm for the scaling formulation consisted of two phases: optimisation and improvement. For the optimisation phase we solved the scaled formulation model using the reformulation descent method, whilst for the improvement phase, we implemented a strategy whose aim is to improve the solution obtained in the optimisation phase, this consist of swapping radii of similar sized circles (and inserting smaller circles if appropriate).

Regarding quality of the results produced for our heuristic for the circular container, comparisons made indicate that in most cases our heuristic results dominated those found in the literature for large and small variation instances. We also tested single formulations (Cartesian formulation only and polar formulation only) and compared the results with the mixed formulation. Results from the test conducted indicated that using a mixed formulation for the packing problem with non-identical circles inside the unit circle container with large variation instances is the most suitable option to obtain more accurate solutions in less time. By contrast, although single formulations did not give good quality results, polar formulation dominated the Cartesian formulation in terms of quality of results and in time spent. Regarding the case of small variation instances and with respect to the quality of results, single Cartesian formulation, single polar formulation and the mixed formulation showed a similar behaviour. With respect to computational time polar formulation gave results three times faster than Cartesian, however the mixed formulation was shown to be an effective strategy to obtain results in considerably less time, indeed the lowest of the three approaches.

Chapter 6

Packing non-identical circles inside different containers

The aim of this chapter is to extend the scaled formulation for the packing problem of non-identical circles presented in Chapter 5 to six different containers. The adaptation to other containers requires minor modifications to the original mathematical model and to the heuristic algorithm. To the best of our knowledge the containers that we consider in this chapter have not been addressed before for the case of non-identical circles, hence we present new results for the small and large variation instances with different containers.

For ease of reference in Section 6.1 we present again the scaled model and the heuristic algorithm considering the unit circle container. Section 6.2 is divided in four, presenting in each section the necessary modifications made to adapt the scaled formulation and the heuristic algorithm to: the rectangular containers consisting of the unit square, two rectangular containers with dimensions ($L = 5$ and $L = 10$, $W = 1$); other shaped containers: the right-angled isosceles triangle, a semicircle and a circular quadrant. In Section 6.3 we present the computational results for the small and large variation instances for all containers and in Section 6.4 we give some conclusions.

6.1 Scaled model and algorithm for non-identical circles

Here, for ease of reference we give again the scaled formulation where we consider the unit circle container and the heuristic algorithm given in Chapter 5.

$$\max \quad \rho \tag{6.1}$$

subject to

$$x_i^2 + y_i^2 \leq (1 - \rho R_i)^2 \qquad \forall i \in C \tag{6.2}$$

$$r_i \leq 1 - \rho R_i \qquad \forall i \in P \tag{6.3}$$

$$(x_i - x_j)^2 + (y_i - y_j)^2 \geq (\rho(R_i + R_j))^2 \qquad \forall (i,j) \in Q \text{ with } i,j \in C \; i < j \tag{6.4}$$

$$(x_i - r_j \cos(\theta_j))^2 + (y_i - r_j \sin(\theta_j))^2 \geq (\rho(R_i + R_j))^2 \quad \forall (i,j) \in Q \text{ with } i \in C \; j \in P \tag{6.5}$$

$$r_i^2 + r_j^2 - 2 r_i r_j \cos(\theta_i - \theta_j) \geq (\rho(R_i + R_j))^2 \qquad \forall (i,j) \in Q \text{ with } i,j \in P \; i < j \tag{6.6}$$

$$-1 \leq x_i \leq 1 \qquad \forall i \in C \tag{6.7}$$

$$-1 \leq y_i \leq 1 \qquad \forall i \in C \tag{6.8}$$

$$0 \leq r_i \leq 1 \qquad \forall i \in P \tag{6.9}$$

$$0 \leq \theta_i \leq 2\pi \qquad \forall i \in P \tag{6.10}$$

$$0 \leq \rho \leq \sqrt{1/\Sigma_{i=1}^{n} R_i^2} \tag{6.11}$$

Let us recall that the objective function (6.1) maximises the scaling factor associated with each radius R_i. Constraints that guarantee that all circles are inside the unit circle container are described by equations (6.2) and (6.3). Equations (6.4), (6.5) and (6.6) are known as the non-overlapping constraints. Equations (6.7) and (6.8) represent the limits for the variables that are in the Cartesian system whilst equations (6.9) and (6.10) represent the limits of the variables that are in the Polar system. Lastly the upper limit

on ρ is represented in equation (6.11).

The modifications to adapt the scaled formulation given in equations (6.1)-(6.11) are related to the boundary of the container considered (6.12) whilst the changes to the heuristic algorithm 6.8 for other types of geometric containers are minor changes in the correction step, which consist of reducing (if needed) the size of each circle which is defined by the common scaling factor and their respective radius to prevent overlaps of any circle with the container equation (6.12) and to avoid overlaps between any two circles equation (6.13). Hence, the value of the maximum scaling factor is determined by equation (6.14).

$$\rho_1 = \min\{(1 - \sqrt{x_i^2 + y_i^2})/R_i | i = 1, ..., n\} \tag{6.12}$$

$$\rho_2 = \min\left\{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}/(R_i + R_j) | i < j \ \forall (i,j) \in Q)\right\} \tag{6.13}$$

$$\rho_{current} = \min\{\rho_1, \rho_2\} \tag{6.14}$$

6.2 Different containers

The scaled formulation model for the packing problem presented in Chapter 5 allowed us to fit non-identical circles without overlaps inside the unit circle container. This approach is now applied to other containers: the unit square, two rectangles with length $L = 5$, $L = 10$ and width $W = 1$, a right-angled isosceles triangle, a semicircle with radius one and a circular quadrant with radius one. This new scaled point of view where the aim is to maximise the scaling factor having fixed size circles and a fixed container is equivalent to minimising the size of the container with fixed size circles.

Without loss of generality, let us say that (x_i, y_i) with $i = 1, ..., n$ is the solution of the scaled problem with a scaling factor ρ_{max} for any container, in particular if we consider the unit square container ($L = W = 1$), then by using $1/\rho_{max}$ as the re-scaling factor

Algorithm 6.8 Reformulation descent for the non-identical circle case

Function $(\rho_{max}, X^*, Y^*) \leftarrow RD(n, replication\_limit, iteration\_limit)$

$t\_rep \leftarrow 0$

repeat

$\rho_{max} \leftarrow 0 \;\; R^{index} \leftarrow [R_1, ..., R_n] \;\; |C| \leftarrow \lfloor n/2 \rfloor \; t \leftarrow 0$ {Initialisation}

Phase 1:

repeat

$(x^0, y^0) \leftarrow$ Randomly generate an initial solution (X, Y)

$C \leftarrow \{1, ..., n\} \setminus P; \;\; P \leftarrow n \setminus C$ {set C and P}

$(x, y, \rho) \leftarrow NLP(x^0, y^0, C, P)$

$\rho_{current} \leftarrow Correction(x^0, y^0, x, y, \rho)$ {correct the radius}

$\rho_{max} \leftarrow \max\{\rho_{max}, \rho_{current}\}$ {update ρ_{max}}

$(X^*, Y^*) \leftarrow (x, y)$ {save coordinates associated with ρ_{max}}

$t \leftarrow t + 1$ {update iteration counter}

until $t = iteration\_limit$

$(X^*, Y^*, \rho_{max}) \leftarrow$ Best solution {best solution from phase 1}

Phase 2:

$(X^*, Y^*, \rho_{max}, R^{index}) \leftarrow Swapping(X^*, Y^*, \rho_{max}, R^{index})$ {ρ_{max} best solution after swap}

repeat

$(x_{best}, y_{best}) \leftarrow Insertion(X^*, Y^*, \rho_{max}, R^{index})$ {best coordinates to insert a small circle}

$(X^*, Y^*) \leftarrow (X^*, Y^*) \cup (x_{best}, y_{best})$

until all circles are being packed

$t\_rep \leftarrow t\_rep + 1$ {update replication counter}

until $t\_rep = replication\_limit$

we obtain $(x_i/\rho_{max}, y_i/\rho_{max})$ as the coordinate solution for a square with dimensions $(L = W = 1/\rho_{max})$.

In a more general way for any rectangular container with dimension (L, W) the re-scaled size would be, length (L/ρ_{max}) and width (W/ρ_{max}). In a similar way the length L of the equal sides of the right-angled isosceles triangle after re-scaling is L/ρ_{max}. The semicircle and right quadrant container have radius one, then as with the unit circle container, the re-scaled size for them is given by $1/\rho_{max}$.

6.2.1 Rectangular containers

For the case of a rectangular container we consider a rectangle of length L and width W with its centre at the origin of the Cartesian plane. Obviously a square has $L = W$ so need not be considered separately here. As mentioned in 2.4.2 the rectangular container has being subject of interests specially for industrial applications as in [17], but none (to the best to our knowledge) address the small and large variation instances that we consider here. In the literature we have found different approaches for the rectangular container with identical or non-identical circles as being of fixed size and the container as being of variable size. Examples of objectives for the rectangular container include:

- for the square (so $L = W$) minimise the length of the side, so minimise L, this also minimises the perimeter and the area of the square
- minimise the perimeter of the rectangle, equivalently minimise $L + W$
- minimise the area of the rectangle, so minimise LW
- regard one dimension of the rectangle as fixed and minimise the other dimension, so for a fixed L minimise W, or for a fixed W minimise L. Problems of this type are often referred to as strip packing problems (SPP) or as the circular open dimension problem (CODP).

With our scaled viewpoint we are addressing a different problem. Namely scale the circles to fit inside the fixed rectangle. There is no direct analogy with any of the other rectangular container ($L \neq W$) objectives listed above. so here we are considering a problem not considered in the literature before.

Modifications with respect to the formulation (which are minor) and those related to the insertion process to deal with our scaled view of rectangular containers are given in detail below.

6.2.2 Formulation modifications

Equations (6.2) and (6.3) ensure that all circles lie inside the container are replaced by

$$\begin{aligned} -L/2 \leq x_i + \rho R_i \leq L/2 \quad -L/2 \leq x_i - \rho R_i \leq L/2, \quad \forall i \in C \\ -W/2 \leq y_i + \rho R_i \leq W/2 \quad -W/2 \leq y_i - \rho R_i \leq W/2 \end{aligned} \tag{6.15}$$

The polar expression equivalent of equation (6.3) is exactly as equation (6.15) but replacing the Cartesian variables by the polar ones. As for the limits on the polar variables equation (6.9) is changed to:

$$0 \leq r_i \leq \sqrt{(L/2)^2 + (W/2)^2} \quad \forall i \in P \tag{6.16}$$

The upper limit on ρ in equation (6.11) becomes $\sqrt{LW/\pi \sum_{i=1}^{n} R_i^2}$.

6.2.3 RD heuristic modifications

With respect to the heuristic during the correction step equation (6.12) is replaced by

$$\rho_1 = \min \{(L/2 - |x_i|)/R_i, (W/2 - |y_i|)/R_i | i = 1, ..., n\} \tag{6.17}$$

which represents the maximum possible scaling factor having regard to the edge of the container and the closest circle centre.

6.2.4 Insertion process modifications

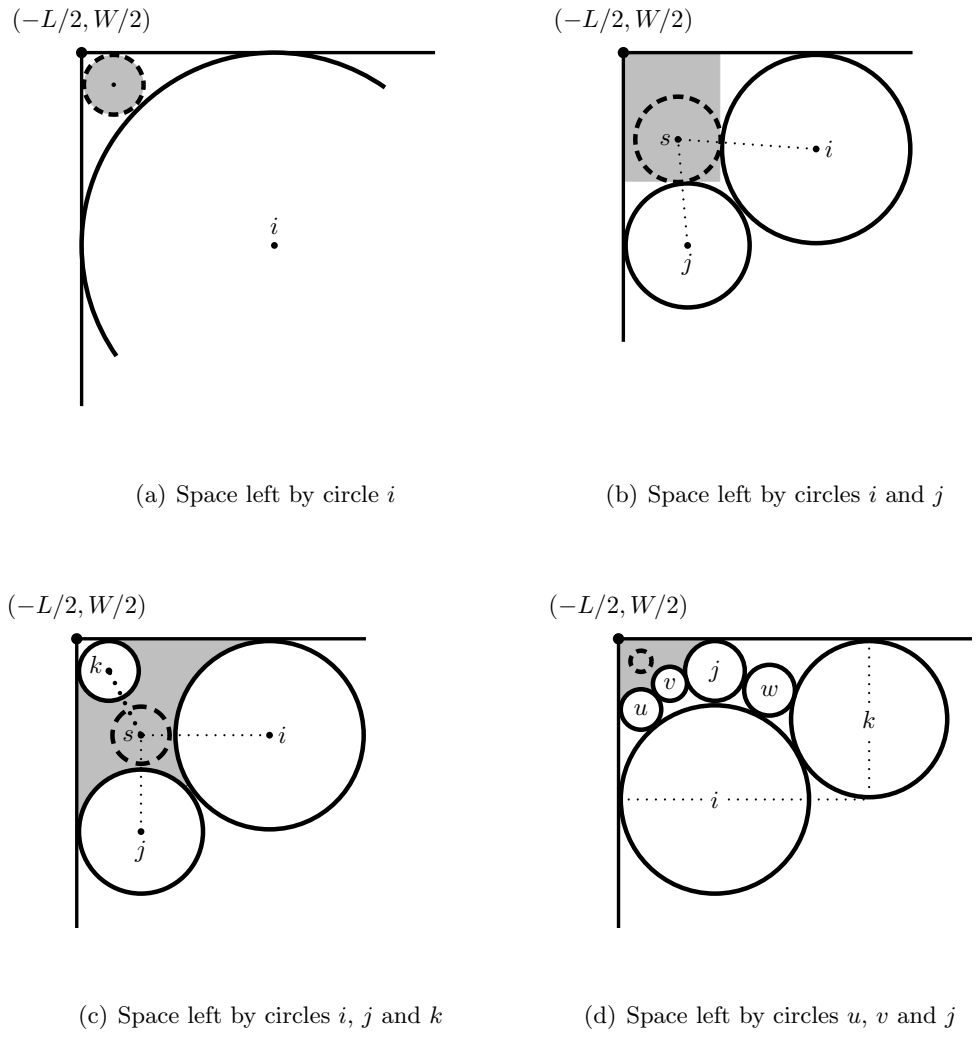
For the insertion process (where we are seeking to insert a new small circle s into an already existing packing) we divided the search for free space into two procedures: one to examine corners and the other to examine the sides of the rectangular container. Both procedures return a list of possible coordinates for the location of the new small circle, from these lists we choose the position that is (if possible) feasible (so without overlaps and inside the container).

6.2.4.1 Corners

While investigating if a corner of a rectangular container represents a feasible location for circle s with radius R_s, (so ρR_s after scaling), we considered three cases: an empty corner where the free space is left by one circle as shown in Figure 6.1(a), an empty corner where the free space is left by two tangent circles as shown in Figure 6.1(b) and a non-empty corner where the free space is left by three circles as shown in Figure 6.1(c). The search around the corners is done one by one and we apply different methods according to the respective situation. After having investigated the four corners we will have a pair of coordinates that represents the best candidate location for circle s in one corner.

For illustration we consider in detail here the upper left corner of the rectangular container, the other three corners are dealt with in a similar fashion. Suppose that the corner under investigation is as depicted in Figure 6.1(a), if it happens to have a big circle but still with enough space to allocate circle s we have coordinates that depend on that corner, here $(\hat{x}, \hat{y}) = (-L/2 + \rho_{max} R_s, W/2 - \rho_{max} R_s)$ as a possible location to position circle s.

If the situation that we are facing is as for the case where there is an empty corner as depicted in Figure 6.1(b), then we build a non-linear system of equations that will be given to the solver to find a feasible solution $(\hat{x}, \hat{y})$ if possible. The elements of this non-linear system are equations (5.15) and (5.16) to guarantee no overlaps between circle



(a) Space left by circle i

(b) Space left by circles i and j

(c) Space left by circles i, j and k

(d) Space left by circles u, v and j

Figure 6.1: Example of the four cases considered to insert a small circle near the corner of a rectangular container

s, circle i and circle j, and equation (6.18) that defines the range for the centre of the new circle.

$$-L/2 + \rho_{max} R_s \leq x \leq x_i - \rho_{max} R_i \quad y_j + \rho_{max} R_j \leq y \leq W/2 - \rho_{max} R_s \tag{6.18}$$

For the case depicted in Figure 6.1(c), we can see two tangent circles (i and j) and one circle (k) in the top left corner, the coordinates of circle s must be found in the grey area, for that we used equations (5.15), (5.16), (6.18) and (6.19). As before equations (5.15)

and (5.16) prevent the overlap of circle s with circles i and j, equation (6.18) restricts the range for the circle centre. Equation (6.19) deals with non-overlap between circles k and s. In equation (6.19) $\rho_{max}R_k$ is the radius of the circle in the corner.

$$(x - x_k)^2 + (y - y_k)^2 - (\rho_{max}R_s + \rho_{max}R_k)^2 \quad \geq \quad 0 \tag{6.19}$$

This forms a non-linear system that will be given to the solver to find (if possible) a feasible solution.

As the total number of circles to be packed increases, more small circles need to be packed in the insertion process (recall in the optimisation phase we only consider 70% of the circles, leaving 30% for the insertion process). In Figure 6.1(d) we consider the case where there are two pairs of circles around the corner, in this case the pairs of circles are (i, j) and (i, k) not considering circles u, v and w because the search is based on circles that are tangent to the container and between themselves. The pair (u, i) fall in the category described in Section 6.2.4.2 below. Investigating these cases around the corner we take the two pairs of circles (i, j) and (i, k) creating a limit on the x-coordinate and on the y-coordinate as denoted by the dotted lines in Figure 6.1(d). If there are circles that are neither i, j nor k near the corner within these dotted lines, as for example circles u, v and w, we search for the nearest pair of circles or the nearest single circle to the corner. If the nearest turns out to be a pair like (v, j) then the case is treated as that depicted in Figure 6.1(c); if the nearest is one circle let us say circle v then it will be tackled as depicted in Figure 6.1(a).

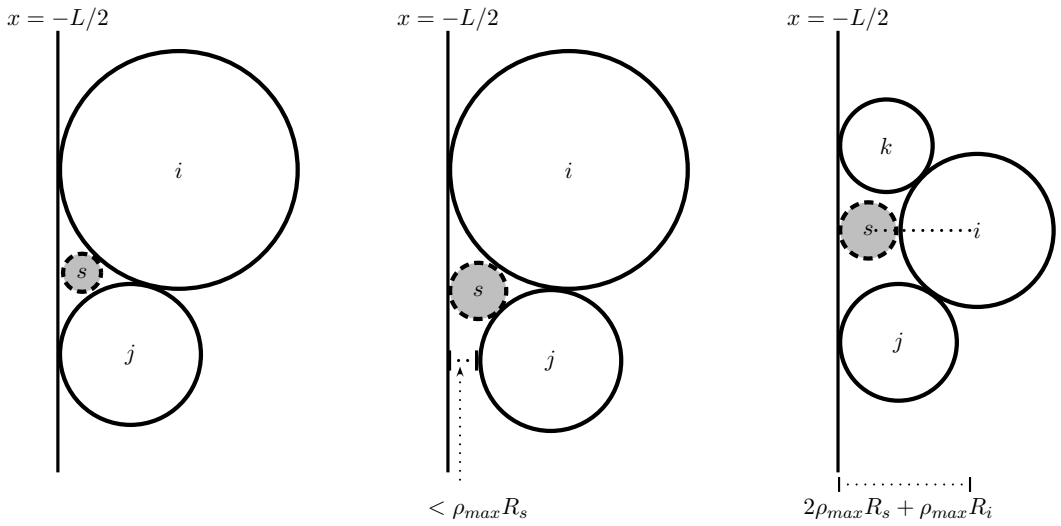
6.2.4.2 Sides

In this section we describe the way the coordinates of circle s are determined on one side of the container. For illustration we consider the left side of the rectangular container, the other three sides are dealt with in a similar manner.

As we can see in Figure 6.2(a) circles i and j are touching the left side of the rectangular container whilst in Figure 6.2(b) we can see that circle j is slightly separated from

the edge of the container. We consider that a circle is close enough to the edge of the container if the distance from its edge to the edge of the container is less than $\rho_{max}R_s$. These two cases are slightly different but will be tackled in the same way. To find the coordinates $(\hat{x}, \hat{y})$ of circle s we need to solve the non-linear system given by equations (5.15) and (5.16) with equation (6.20).

$$x - \rho_{max}R_s \quad \geq \quad -L/2 \tag{6.20}$$



(a) Space left by circles i and j

(b) Space left by circles i and j

(c) Space left by circles i, j and k

Figure 6.2: Example of the empty space left by two tangent circles to allocate a smaller circle on a side of a rectangular container

For the last case let us consider Figure 6.2(c), here we are interested to know if the distance between circle i and the edge of the container is big enough ($2\rho_{max}R_s$) to insert circle s, as we are considering the left side of the container then the coordinates for the centre of circle s would be given by $\hat{x} = -L/2 + \rho_{max}R_s$ and $\hat{y} = y_i$. If the area happens to be smaller we still consider coordinates $(\hat{x}, \hat{y})$ as defined previously to form part of the current solution but taken now as the initial solution for the solver.

6.2.5 Triangular container

For the right-angled isosceles triangle the modifications for the formulation and heuristic algorithm are related to boundary constraints of the container. Additionally we present an adaptation to the insertion process for this particular container, although we used those described for the rectangle in 6.2.4.2 that concern to the sides of the container, we incorporated here one more strategy to insert small circles around the diagonal of the triangular container.

6.2.5.1 Formulation modifications

Regarding the formulation changes equation (6.2) is replaced by equation (6.21), whilst equation (6.3) is replaced by the polar equivalent to equation (6.21).

$$x_i + y_i + \sqrt{2}\rho R_i \leq L \qquad \forall i \in C \tag{6.21}$$

To fully ensure that all circles are inside the triangle container we need to add equations (6.22) with its polar equivalent equation.

$$x_i \geq \rho R_i \qquad y_i \geq \rho R_i \qquad \forall i \in C \tag{6.22}$$

With respect to the variable limits for circles expressed in the Cartesian system, equations (6.7) and (6.8) are deleted, whilst for circles expressed in the polar system we keep constraints as in equations (4.19). The upper limit for variable ρ is replaced by $L/\sqrt{2\pi \sum_{i=1}^{n} R_i^2}$.

6.2.5.2 RD heuristic modifications

Equation (6.12), to ensure that the circles are fully inside the container, is replaced by

$$\rho_1 = \min\left\{x_i/R_i, y_i/R_i, (L - x_i - y_i)/(\sqrt{2}R_i) \mid i = 1, ..., n\right\} \tag{6.23}$$

6.2.5.3 Insertion process modifications

The modifications needed for this container for the insertion process are associated with those described in Section 6.2.4.2 which are related to the sides of the container. Hence, here we present the modifications related to the diagonal of the triangle. To illustrate how we determine if it is possible to locate small circles around the diagonal of the triangle let us consider Figure 6.3.

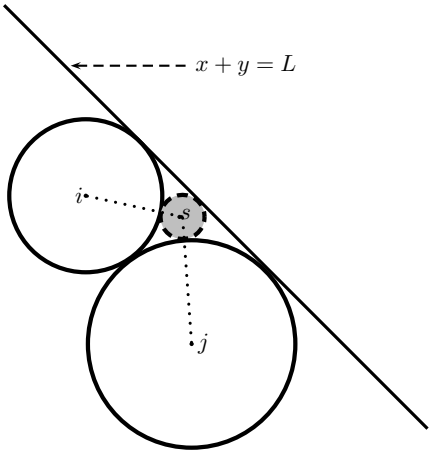


Figure 6.3: Inserting small circles around the diagonal of the right-angled isosceles triangle

Investigating for a feasible location for circle s with radius R_s, (so ρR_s after scaling) around the diagonal of the triangle, we look for all circles that are tangent to the diagonal, from those tangent circles we form pairs of circles which as well as tangent to the diagonal of the container are tangent circles one to another, as circles i and j in Figure (6.3). We consider the pair of circles (i, j) to form the non-linear system given in equations (5.15), (5.16) and equation (6.24) to find a centre coordinate (x, y) for circle s with radius R_s.

$$x + y + \sqrt{2}\rho_{max} R_s \quad \leq \quad L \tag{6.24}$$

6.2.6 Semicircular container

The semicircular container has not been considered in the literature before (to the best of our knowledge) for the small nor the large variation instances that we are addressing, hence here we present the modification to the scaled formulation and the heuristic.

6.2.6.1 Formulation modifications

For this container we add equation (6.25) with its equivalent polar form.

$$y_i \geq \rho R_i \qquad \forall i \in C \tag{6.25}$$

Regarding the limits of the centres expressed in Cartesian coordinate system, equation (6.8) is replaced by $0 \leq y_i \leq 1 \ \forall i \in C$, whilst for centres expressed in polar coordinate system equation (6.10) is modified as $0 \leq \theta_i \leq \pi$. The upper limit for variable ρ is replaced by $\sqrt{1/2 \sum_{i=1}^{n} R_i^2}$.

6.2.6.2 RD heuristic modifications

Equation (6.12), to ensure that the circles are fully inside the container, is replaced by

$$\rho_1 = \min \left\{ y_i/R_i, (1 - \sqrt{x_i^2 + y_i^2})/R_i \mid i = 1, ..., n \right\} \tag{6.26}$$

6.2.6.3 Insertion process modifications

Modifications to the insertion process for the semicircular container have been already described: inserting small circles around the curve of the semicircle follow the same strategy as for the unit circle container detailed in Section 5.3.5, whilst for the bottom side of the semicircle we follow the strategy described in Section 6.2.4.2.

6.2.7 Circular quadrant container

The circular quadrant as well as the semicircular container has not been considered before for the packing problem with non-identical circles. Here we considered the upper right circular quadrant. The modifications to the scaled formulation and the heuristic algorithm to have the circular quadrant with radius one as container will be given below.

6.2.7.1 Formulation modifications

For this container we add equation (6.27) with its equivalent polar form.

$$x_i \geq \rho R_i \qquad y_i \geq \rho R_i \qquad \forall i \in C \tag{6.27}$$

Regarding the limits of the centres expressed in Cartesian coordinate system, equations (6.7) and (6.8) are replaced by $0 \leq x_i \leq 1$ and $0 \leq y_i \leq 1$, both $\forall i \in C$ whilst for centres expressed in polar coordinate system equation (6.10) is modified as $0 \leq \theta_i \leq \pi/2$. The upper limit for variable ρ is replaced by $\sqrt{1/4 \sum_{i=1}^{n} R_i^2}$.

6.2.7.2 RD heuristic modifications

Equation (6.12), to ensure that the circles are fully inside the container, is replaced by

$$\rho_1 = \min \left\{ x_i/R_i, y_i/R_i, (1 - \sqrt{x_i^2 + y_i^2})/R_i \mid i = 1, ..., n \right\} \tag{6.28}$$

6.2.7.3 Insertion process modifications

Modifications to the insertion process for the circular quadrant container are similar to those for the semicircle: inserting small circles around the curve of the circular quadrant follow the strategy as for the unit circle container given in Section 5.3.5, whilst for the bottom and left side of the circular quadrant we adopt the strategy described in Section 6.2.4.2.

6.3 Computational results

The instances presented here consider six different containers; the unit square (so a rectangle of length $L = 1$ and width $W = 1$), a rectangle of length $L = 5$ and width $W = 1$, a rectangle of length $L = 10$ and width $W = 1$, a right-angled isosceles triangle, a semicircle with radius one and a circular quadrant with radius one. For these six containers we address the case when the radii of the circles to be packed are defined by $R_i = i$ and $R_i = 1/\sqrt{i}$ for all $i = 1, ..., n$, giving the best solution obtained by our heuristic. For the rectangular cases we present results for instances where length $L = 5$ and $L = 10$ and width $W = 1$ considering $R_i = \sqrt{i}$ for all $i = 1, ..., n$. Instances of the type $R_i = i$ we treat as large variation instances whilst those with $R_i = 1/\sqrt{i}$ or $R_i = \sqrt{i}$ we treat as small variation instances.

The hardware used is an Intel(R) Core(TM) i5-2500 3.30 GHz CPU with 4.00 GB. The algorithm was coded in MatLab 7.9.0 using SNOPT [18, 35] as the non-linear solver.

6.3.1 Rectangular containers

Recall that, for rectangular containers, our scaled view of the unequal circles packing problem leads to new problems that have not been considered previously in the literature. In this section we present results for scaled problems involving three different rectangular containers: the unit square, a rectangle of length $L = 5$ and width $W = 1$ and a rectangle of length $L = 10$ and width $W = 1$. Here we again consider $R_i = i$ (so large variation instances) and $R_i = 1/\sqrt{i}$ (so small variation instances) as in Table 5.1 and Table 5.2.

We found that the small variation instances ($R_i = 1/\sqrt{i}$) with a rectangle of length $L = 5$ and width $W = 1$ always (for $n \leq 35$) gave a solution $\rho_{max} = 0.5$. The reason for this is explained in the next paragraph. So here we consider small variation instances of the form $R_i = \sqrt{i}$ for the rectangle $L = 5$, $W = 1$. A similar situation is encountered for the rectangle with dimensions ($L = 10, W = 1$), hence the small variation instances for

this container is with $R_i = \sqrt{i}$.

When $R_i = 1/\sqrt{i}$ the largest circle has radius after scaling of $\rho R_1 = \rho$ whilst the smallest circle has radius $\rho/\sqrt{n}$. As the objective is to find the maximum scaling factor ρ to fit the circles without overlaps, for the case $L = 5$, $W = 1$ ρ will be bounded by the largest circle and the width of the rectangular container, hence as the largest circle has radius 1 (diameter 2) and $W = 1$ then with $\rho = 0.5$, the largest circle with scaled radius 0.5 is inside and touching the top and bottom sides of the container. Hence $\rho = 0.5$ will pack the largest circle into the rectangle $L = 5$, $W = 1$. Computational experience indicated that for $n \leq 35$ we could also pack the other circles with $\rho = 0.5$. If we wish to find the value for n for which we definitely cannot pack the n circles into the rectangle with $\rho = 0.5$ then, from area considerations, n must satisfy $\sum_{i=1}^{n} \pi(0.5R_i)^2 > LW$ i.e. $\sum_{i=1}^{n} \pi(0.5)^2(1/\sqrt{i})^2 = 0.25\pi \sum_{i=1}^{n}(1/i) > LW$. This is a harmonic series and it is easy to verify numerically that when $L = 5$, $W = 1$ we need $n \geq 327$. Such a large value of n would be outside the effective computational range of our RD heuristic and so we do not consider the case $R_i = 1/\sqrt{i}$ for the rectangle $L = 5$, $W = 1$ here. Similarly for $L = 10$, $W = 1$ we also do not consider the case $R_i = 1/\sqrt{i}$.

In Table 6.1 we present the best results from our RD heuristic after five replications. Each replication consists of 50 iterations for the unit square, whilst for the rectangular containers with $L = 5$, $W = 1$ and $L = 10$, $W = 1$ we (for computational reasons) halved the number of iterations from 50 to 25.

The instances considered range from $n = 5$ to $n = 35$ circles. The results presented in Table 6.1 are presented in terms of ρ_{max} which is the maximum common scaling factor, such that multiplying each radius by ρ_{max} enables all circles to be packed into the container considered. In Figure 6.4 we present a picture that shows the best results obtained by our RD heuristic for the rectangular containers with $n = 35$ circles. Considering the average times at the foot of Table 6.1 and Table 6.2 it is clear that our RD heuristic is computationally more demanding when we have the rectangular container with $L = 10$

and $W = 1$ than when we have the the rectangular container with $L = 5$ and $W = 1$ or the unit square container.

Because the scaled view for rectangular containers has not been considered in the literature previously we cannot make any comparison between the results in Table 6.1 and previous work in the literature. However we believe that the scaled view is a natural way to view the problem of packing unequal circles into containers (of any type) and expect that later work in the literature will adopt the same view and so future workers will be comparing their results against the ones we have presented in Tables 6.1 and 6.2.

Table 6.1: RD results for the large variation instances $R_i = i$ for the unit square and two rectangles of length $L = 5$, $L = 10$ and width $W = 1$

n	Best result found ρ_{max} Unit square $L = W = 1$	Total time in minutes	Best result found ρ_{max} Rectangle $L = 5$ $W = 1$	Total time in minutes	Best result found ρ_{max} Rectangle $L = 10$ $W = 1$	Total time in minutes
5	0.06408909	61.35	0.10000000	25.29	0.10000000	21.71
6	0.05148561	63.91	0.08333333	42.96	0.08333333	36.99
7	0.04198350	71.10	0.07142857	66.75	0.07142857	62.06
8	0.03435803	68.24	0.06250000	71.45	0.06250000	64.27
9	0.02962585	85.54	0.05555556	103.52	0.05555556	95.15
10	0.02591988	103.81	0.05000000	138.56	0.05000000	141.45
11	0.02237621	125.08	0.04545455	184.89	0.04545455	198.31
12	0.01991135	121.35	0.04166667	176.87	0.04166667	206.60
13	0.01775996	147.63	0.03804136	218.81	0.03846154	271.91
14	0.01600778	171.13	0.03410353	273.82	0.03571429	350.14
15	0.01453245	204.66	0.03065862	339.24	0.03333333	461.34
16	0.01324257	200.47	0.02804934	353.39	0.03125000	454.01
17	0.01213436	238.52	0.02556902	434.54	0.02941176	573.40
18	0.01116368	279.11	0.02360413	497.38	0.02777778	708.75
19	0.01029409	318.65	0.02193570	587.17	0.02631579	863.57
20	0.00955011	316.77	0.02049985	598.45	0.02500000	872.58
25	0.00688601	598.91	0.01490721	1167.88	0.02000000	1662.87
30	0.00529005	1024.13	0.01151872	1959.20	0.01563915	3296.32
35	0.00421150	1665.16	0.00922810	3298.13	0.01245317	5557.62
Average		308.71		554.65		836.79

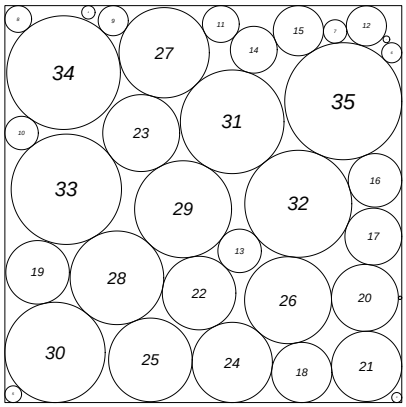
Table 6.2: RD results for the small variation instances for the unit square and two rectangles of length $L = 5$, $L = 10$ and width $W = 1$

(a) Results when $R_i = 1/\sqrt{i}\ \forall i = 1, ..., n$

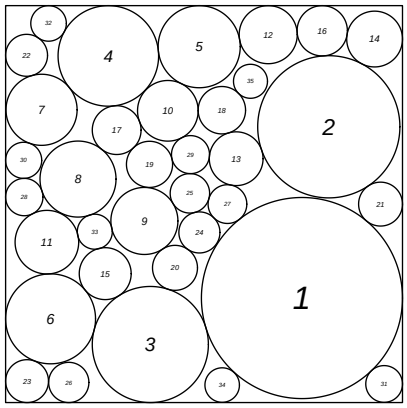
n	Best result found ρ_{max} Unit square $L = W = 1$	Total time in minutes
5	0.32330862	17.32
6	0.30882677	21.34
7	0.30571049	24.95
8	0.30270079	28.87
9	0.30048241	34.29
10	0.29769203	41.66
11	0.29380771	48.91
12	0.28859081	58.79
13	0.28574260	70.34
14	0.28244379	84.15
15	0.28115079	99.87
16	0.27906616	118.61
17	0.27518822	140.17
18	0.27413312	165.28
19	0.27190253	191.76
20	0.27056353	223.22
25	0.26257532	444.55
30	0.25828123	828.73
35	0.25330659	1466.18
Average		216.26

(b) Results when $R_i = \sqrt{i}\ \forall i = 1, ..., n$

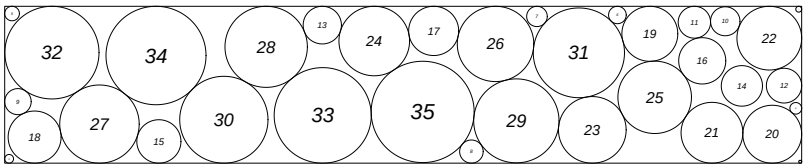
n	Best result found ρ_{max} Rectangle $L = 5\ W = 1$	Total time in minutes	Best result found ρ_{max} Rectangle $L = 10\ W = 1$	Total time in minutes
5	0.22360680	28.39	0.22360680	28.20
6	0.20412415	39.06	0.20412415	44.23
7	0.18898224	52.64	0.18898224	72.09
8	0.17677670	58.44	0.17677670	103.47
9	0.15820579	74.93	0.16666667	145.01
10	0.14202479	94.98	0.15811388	189.79
11	0.12973535	120.15	0.15075567	237.79
12	0.11970014	146.63	0.14433757	294.21
13	0.11083253	181.17	0.13867505	343.42
14	0.10351509	218.03	0.13363062	397.54
15	0.09723726	263.12	0.12909944	444.69
16	0.09235499	313.29	0.12500000	477.04
17	0.08733291	374.53	0.12126781	482.40
18	0.08345285	446.17	0.11461741	577.36
19	0.07943738	524.95	0.10989643	697.41
20	0.07617540	617.6	0.10273367	846.55
25	0.06224501	1341.57	0.08205458	1955.45
30	0.05193559	2668.18	0.06950574	3687.23
35	0.04464354	4458.87	0.06059512	5408.43
Average		632.77		864.86



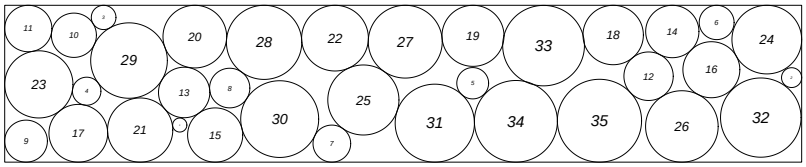
(a) Unit square $R_i = i$



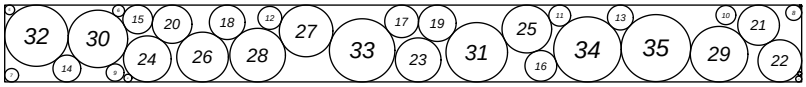
(b) Unit square $R_i = 1/\sqrt{i}$



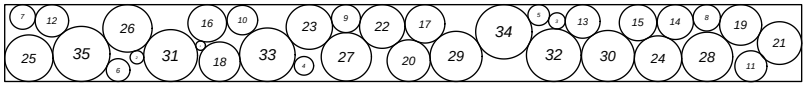
(c) Rectangle $L = 5$, $W = 1$, $R_i = i$



(d) Rectangle $L = 5$, $W = 1$, $R_i = \sqrt{i}$



(e) Rectangle $L = 10$, $W = 1$, $R_i = i$



(f) Rectangle $L = 10$, $W = 1$, $R_i = \sqrt{i}$

Figure 6.4: Circle packing problem with $n = 35$ non-identical circles

6.3.2 Other containers

Here we present the computational results that correspond to a new set of instances for the right-angled isosceles triangle, the semicircle and circular quadrant container. For each container we considered the large and the small variation instances ($R_i = i$ and $R_i = 1/\sqrt{i}$). The number of circles considered in each set range from $n = 5, 6, ...20, 25, 30$ and 35. The results were produced with our RD heuristic after 5 replications, where each replication consisted of 25 iterations.

In Table 6.3 we present the results for the large variation instances, that is, when the radii is defined as $R_i = i$ $i = 1, ..., n$ (before scaling) and the total computation time given in minutes for the three containers considered. In Table 6.3 we can see from the last row that the right-angled isosceles triangle is the container that on average takes most computation time to solve large variation instances with an average total time of 104.49 min, followed by the semicircular container with an average total time of 60.06 min. and lastly the circular quadrant with an average total time of 58.63 min.

In a similar way, Table 6.4 is as Table 6.3 but for the small variation instances considering the same three containers. Here again, the right-angled isosceles triangle is the container that requires more time to solve a small variation instance, from the bottom row in Table 6.4, this container on average takes 119.66 min, whilst the semicircular and circular quadrant containers take an average total time of 106.31 min and 102.68 min respectively.

For illustrative purpose we give Figure 6.5 that shows the packing of 35 non-identical circles for large variation instances for the right-angled isosceles triangle, a semicircle and a circular quadrant container in Figures 6.5(a), 6.5(c) and 6.5(e) respectively, whilst an example of small variation instances are in Figures 6.5(b), 6.5(d) and 6.5(f).

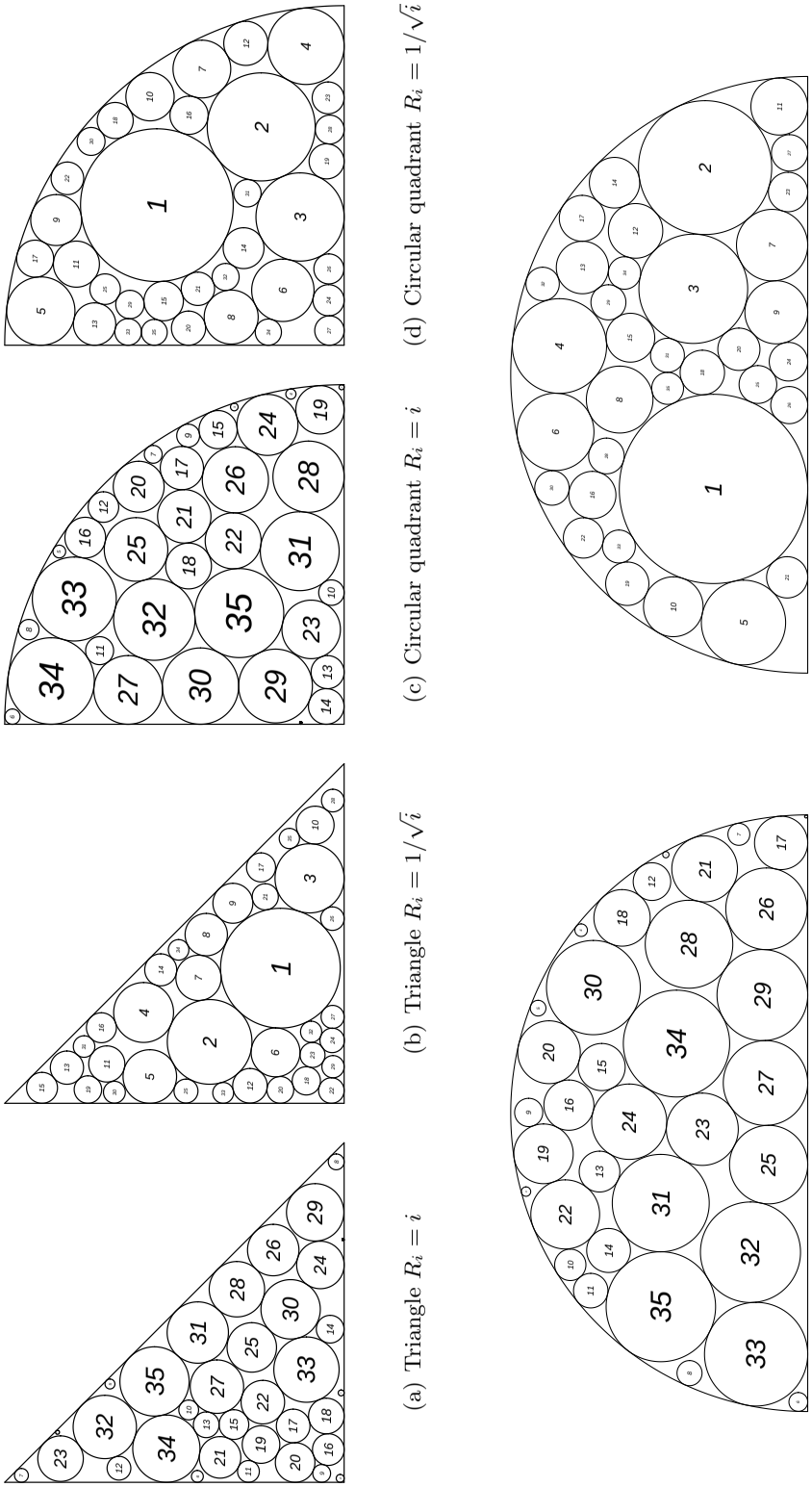
As for the rectangular container, to the best of our knowledge there is not yet work considering the large and/or the small variations instances with these containers, hence it was not possible to make any comparisons for both.

Table 6.3: RD heuristic results for the triangle, the semicircle and the circular quadrant when $R_i = i$ $i = 1, ..., n$

n	Best result found ρ_{max} Triangle	Total time in min	Best result found ρ_{max} Semicircle	Total time in min	Best result found ρ_{max} Circular quadrant	Total time in min
5	0.04322180	19.54	0.08276031	4.23	0.05760692	3.97
6	0.03402663	24.39	0.06509192	5.82	0.04483669	5.57
7	0.02799637	17.72	0.05251289	7.85	0.03721518	7.39
8	0.02362081	20.82	0.04402540	7.99	0.03085453	7.95
9	0.02044733	21.28	0.03699919	10.65	0.02633990	10.08
10	0.01751620	22.15	0.03221533	13.75	0.02284130	13.11
11	0.01540367	25.78	0.02825222	17.59	0.01995210	16.65
12	0.01359076	26.36	0.02510088	17.94	0.01762054	17.42
13	0.01220424	30.01	0.02232198	22.09	0.01571118	21.55
14	0.01096585	35.12	0.02000427	27.54	0.01420988	26.55
15	0.00995738	42.28	0.01820520	33.86	0.01282254	31.99
16	0.00906386	43.80	0.01655209	34.67	0.01172732	33.18
17	0.00831046	54.37	0.01510383	42.11	0.01077823	40.47
18	0.00767919	66.25	0.01394521	49.76	0.00985276	47.99
19	0.00707264	80.75	0.01290924	59.56	0.00913019	57.03
20	0.00660630	84.51	0.01205116	62.57	0.00848638	58.27
25	0.00474646	192.00	0.00861995	119.84	0.00609799	119.90
30	0.00365469	402.23	0.00662500	219.63	0.00470227	224.49
35	0.00291360	775.91	0.00526477	383.78	0.00374555	370.47
Average		104.49		60.06		58.63

Table 6.4: RD heuristic results for the triangle, the semicircle and the circular quadrant when $R_i = 1/\sqrt{i}$ $i = 1, ..., n$

n	Best result found ρ_{max} Triangle	Total time in min	Best result found ρ_{max} Semicircle	Total time in min	Best result found ρ_{max} Circular quadrant	Total time in min
5	0.20000000	21.51	0.40102148	5.22	0.28920238	5.06
6	0.16666667	18.84	0.39455239	6.64	0.28048620	6.77
7	0.14285714	17.05	0.38354371	8.60	0.27389807	8.89
8	0.12500000	18.33	0.37892907	10.91	0.26977048	10.74
9	0.11111111	19.76	0.37528528	13.80	0.26721593	13.54
10	0.10000000	23.35	0.36933870	17.11	0.26344525	16.93
11	0.09090909	27.23	0.36423389	21.09	0.25909024	20.96
12	0.08333333	32.77	0.36091472	25.94	0.25585662	25.74
13	0.07692308	40.26	0.35905701	31.65	0.25324214	31.44
14	0.07142857	46.87	0.35340477	38.26	0.25120939	37.94
15	0.06666667	55.67	0.35041621	46.14	0.24886577	45.52
16	0.06250000	66.44	0.34764393	55.11	0.24675495	54.32
17	0.05882353	76.85	0.34577669	65.37	0.24342555	64.52
18	0.05555556	91.83	0.34257623	77.38	0.24291361	76.27
19	0.05263158	104.62	0.34046525	91.29	0.24124194	89.47
20	0.05000000	121.03	0.33808797	107.09	0.23973491	104.80
25	0.04000000	248.92	0.32990049	224.41	0.23267521	215.85
30	0.03333333	456.78	0.32341164	423.06	0.22829197	405.29
35	0.02857143	785.38	0.31721067	750.74	0.22495966	716.81
Average		119.66		106.31		102.68



(a) Triangle $R_i = i$ (b) Triangle $R_i = 1/\sqrt{i}$ (c) Circular quadrant $R_i = i$ (d) Circular quadrant $R_i = 1/\sqrt{i}$

(e) Semicircle $R_i = i$ (f) Semicircle $R_i = 1/\sqrt{i}$

Figure 6.5: Circle packing problem with $n = 35$ non-identical circles

6.4 Conclusions

In this chapter we extended the new scaled formulation model for packing non-identical circles originally given for the unit circle container in the previous chapter to other containers. The containers considered are: the unit square, a rectangle with length $L = 5$ and width $W = 1$, a rectangle with length $L = 10$ and width $W = 1$, a right-angled isosceles triangle, a semicircle with radius one and a circular quadrant with radius one. For all these containers we considered a set of $n = 5, 6, ..., 20, 25, 30, 35$ non-identical circles for the small variation instances and the large variation instances. We described in detail the appropriate modification to the mathematical model and to the heuristic algorithm.

To the best of our knowledge, there is no work that addresses the packing problem with non-identical circles for other containers than the circular one with which we can compare our results. Some problems found in the literature although address the packing problem with rectangular containers are not equivalent to the problem we are presenting in this thesis. Hence, we are considering a new problem and the results from the large and small variation instances represent a contribution for the six containers considered here.

Chapter 7

MINLP through FSS

In this chapter we present two implementations of formulation space search (FSS) to solve Mixed-Integer Non-linear Programming problems, denoted as MINLP. We focus on problems where the integer variables are zero-one, so binary variables. In Section 7.1 we give a brief overview on the applications where we can find MINLP problems, we present their general formulation model. Section 7.2 presents the reformulation proposed whilst in Section 7.3 we describe our first approach implementing FSS to solve MINLP problems denoted as FSS-MINLP-1, we give the implementation used and draw conclusions over limited computational results. this led us to rethink our approach and to consider a different non-linear solver hence, Section 7.4 presents our second and more successful approach denoted FSS-MINLP-2. Finally in Section 7.5 we end the chapter with conclusions.

7.1 MINLP problem

Mixed-integer non-linear problems (MINLP's) frequently appear in industry, some of the applications are: process flow sheets, portfolio selection, batch processing in chemical engineering, optimal design of gas or water transmission networks [65]. MINLP problems are by nature challenging as they combine discrete and continuous variables and are non-

linear. Problems of this type have been subject of the development of software capable to solve them, in [13] the authors give a descriptive summary of the available solvers such as: αBB, αECP, AOA, BARON, BNB, BONAMI, COUENNE, DICOPT, FICO Xpress-SLP, FILMINT, FMINCONSET, KNITRO, LAGO, LINDOAPI, LOGMIP, MIDACO, MILANO, MINLP_BB, MISQP, MOSEK, OQNLP, SBB and SCIP along with their compatibility with modelling languages such as AIMMS, AMPL, GAMS, and NEOS. One difficulty for researchers into MINLP is that, as this list demonstrates, there are so many possible software packages which can be used. Which one might be best for any particular MINLP is a very open question.

In [22] Grossmann gives an overview of the methods that have been more studied and implemented for MINLP problems such as: branch and bound, outer approximation, generalized benders and extended cutting plane methods. In [11] the authors present the algorithms that were implemented and resulted in the creation of an open-source environment called COIN-OR.

Literature related with mixed-integer non-linear problems is abundant, here we only focused on those with a general perspective, however there are several works addressing the particular problems found in industry that are being tackled as MINLP problems.

The general form of a mixed-integer (zero-one) non-linear problem is

$$\begin{aligned} \min \quad & c(x,y) && (7.1)\\ \text{subject to} \quad & A(x,y) \leq b && (7.2)\\ & x \quad \text{zero-one} && (7.3)\\ & y \quad \text{continuous} && (7.4) \end{aligned}$$

Objective (7.1) describes the aim of the problem which is minimize function c that depends on zero-one variables (x) and/or continuous variables (y). Equation (7.2) represents the constraints. Let us note that c and A are general functions (linear or non-linear) of the zero-one variables x and the continuous variables y. Let us consider the number

of zero-one variable be n and be denoted as x_i $i = 1, ..., n$. Expressions (7.3) and (7.4) represent the limits on zero-one variables and continuous variables respectively.

7.2 FSS formulation for MINLP problems

To solve mixed-integer (zero-one) non-linear programming problems we propose an approach that is based on formulation space search. As we explained in section 2.5 formulation space search consists of solving a problem using equivalent formulations of the original problem that will allow escape from stationary points thus increasing our chances to obtain a better solution.

The basis of the approach presented here considers an equivalent formulation to the original problem.This is based on

$$\sum_{i=1}^{n} x_i(1 - x_i) = 0 \tag{7.5}$$

$$0 \leq x_i \leq 1 \quad i = 1, ..., n \tag{7.6}$$

In essence this single non-linear equality, plus the relaxation of the zero-one requirement to allow x_i to be continuous will automatically ensure that the x_i actually assume zero-one values in the solution. Although this reformulation of the problem is descriptively accurate, and enforces variables x_i to be equal zero or one, equation 7.5 may impose a harder condition to be satisfied, hence we have adopted a more flexible version of equation 7.5 to be the framework of our first approach presented in Section 7.3.

7.3 First approach FSS-MINLP-1

The basic approach involves decomposing equation (7.5) into different constraints, each one of these constraints will be converted into inequalities with right hand side δ_k, this will make the new constraint more easily satisfied.

Before presenting the model let us consider some notation:

- K represents the number of sets we divide the zero-one variables into, having K mutually exclusive sets
- $P_k = \{i|\text{variable } i \in \text{set } k\}\ k \in \{1, ..., K\}$
- α a convergence factor
- T is the iterative counter

The general model for this approach is presented in equations (7.7)-(7.11)

$$\begin{aligned}
\min \quad & c(x, y) && && (7.7)\\
\text{subject to} \quad & A(x, y) \leq b && && (7.8)\\
& \sum_{i \in P_k} x_i(1 - x_i) \leq \delta_k && k = 1, ..., K \quad \text{if} \quad T \neq 1 && (7.9)\\
& 0 \leq x_i \leq 1 && i = 1, ..., n && (7.10)\\
& [x_i], y \quad \text{continuous} && && (7.11)
\end{aligned}$$

the explicit definition of δ_k will be given in detail in Section 7.3.1.

7.3.1 Algorithm

The pseudocode of the algorithm proposed to implement the FSS approach to solve mixed-integer non-linear problems is presented in Algorithm 7.9. The general idea of the FSS approach for mixed-integer zero-one non-linear programming problems is to solve in an iterative way continuous non-linear problems that in every iteration force the currently continuous variables x_i to approach zero-one through equation 7.9.

The algorithm starts with randomly assigning the zero-one variables to a set k (where $k \in \{1, ..., K\}$) where $P_k = \{i|\text{variable } i \in \text{set } k\}\ k = 1, ..., K$. At this point we relax the zero-one variables to continuous variables in the interval $[0, 1]$.

For the iteration process we solve the non-linear problem represented with equations (7.7)-(7.11), this process is denoted in the pseudocode as $NLP(x, y)$, here let us notice that the additional constraints are not considered in iteration one ($T = 1$), that is, for iteration one we have maximum flexibility in deciding values for the zero-one variables $[x_i]$. Once we have obtained a solution from the continuous formulation we record the values of the zero-one variable $[x_i]$ in the current solution using $X_i = x_i \; i = 1, ..., n$. Clearly the X_i may well be continuous in the previous solution arrived at, hence we round $X_i \; i = 1, ..., n$ to its nearest integer $\{0, 1\}$ values giving $[R_i]$. Fixing this integer values to the zero-one variables, we resolve to seek continuous variables that satisfy the constraints (7.12)-(7.14). This problem in the pseudocode is denoted as $NLP(R, y)$.

$$\min \quad c(R, y) \tag{7.12}$$

$$\text{subject to} \quad A(R, y) \leq b \tag{7.13}$$

$$y \quad \text{continuous} \tag{7.14}$$

Achieving feasible solution for MINLP problems may be difficult, hence if a feasible solution from constraints (7.12)-(7.14) is obtained then we have a feasible solution to the mixed-integer zero-one non-linear problem. The iterative process finishes when $\sum_{k=1}^{K} \delta_k \leq 10^{-5}$, if this condition has not been met we can proceed to the next iteration with a different space search by changing both P_k and $\delta_k \; k = 1, ..., K$. We randomly assign each of the zero-one variables to a set k as before. To change δ_k we know that given the current continuous values X_i for the zero-one variables the current left-hand side of the subset sum constraints is $\sum_{i \in P_k} X_i(1 - X_i)$. Hence set $\delta_k = [\sum_{i \in P_k} X_i(1 - X_i)]/\alpha$ with $k = 1, ..., K$. Here we divide the current right-hand side of the subset sum constraints by the factor α at each iteration. The larger the value of α the quicker we force this right-hand side down towards zero.

Algorithm 7.9 FSS-MINLP-1 for mixed-integer non-linear problems pseudocode

Function $(x, y) \leftarrow$ FSS-MINLP-1($replication\_limit$)

$t \leftarrow 0$

repeat

Initialisation: $K \leftarrow \frac{n}{3}$ $\alpha \leftarrow 2$ $T \leftarrow 1$

Randomly assign each zero-one variables to a set k

while not termination condition **do**

$(x, y) \leftarrow NLP(x, y)$ {solves a continuous problem}

$X_i \leftarrow x_i$ {record current cont sol}

$R_i \leftarrow 0$ if $X_i \leq 0.5$, $R_i = 1$ otherwise {rounding to nearest integer}

$y \leftarrow NLP(R, y)$ {solves a continuous problem}

if $\sum_{k=1}^{K} \delta_k \leq 10e^{-5}$ **then**
STOP

else

$T \leftarrow T + 1$ {update iteration counter}

end if

change P_k and δ_k with $k = 1, ..., K$

$\delta_k \leftarrow [\sum_{i \in P_k} X_i(1 - X_i)]/\alpha$ {set δ_k to the new constraints}

end while

until $t = replication\_limit$

7.3.2 Three variants of our approach

In order to increase our chances to reach the best-known solution with our heuristic we have adopted three variants of the formulation described in equations (7.7)-(7.11), the algorithm used is that given in pseudocode 7.9.

Variant 1. The first formulation variant consists of modifying the objective func-

tion leaving the set of constraints as in the original problem, essentially we consider a penalty function with a μ parameter. The formulation variant 1 is represented with equation (7.15) as the objective function and with equations (7.8)-(7.11) as the set of constraints.

$$\min c(x,y) + \mu \sum_{i=1}^{n} x_i(1-x_i) \tag{7.15}$$

The idea for this formulation is to avoid imposing a subset of constraints that may well be more difficult to be satisfied, instead we used a big enough parameter μ, hence when in the objective function solution the part with the parameter is equal to zero, then we know that we have found a feasible solution of the original problem.

Variant 2 and Variant 3. For numerical reasons related to rounding errors we have adopted as Variant 2 a modification of the formulation given in equations (7.8)-(7.11) by considering equation (7.9) as

$$\sum_{i=1}^{n} |x_i||(1-x_i)| \leq \delta_k \quad k = 1, ..., K. \tag{7.16}$$

In a similar way Variant 3 is a modification of Variant 1 where the objective function given in equation (7.15) is now considered as

$$\min c(x,y) + \mu \sum_{i=1}^{n} |x_i||(1-x_i)| \tag{7.17}$$

7.3.3 Computational results

Results presented in this section were produced using an Intel Core 2 pc (2.26GHz, 4GB RAM). Our heuristic was coded in MATLAB 7.0 (using Windows XP) and as a subroutine we used the non-linear optimisation solver SNOPT [18, 35].

As a first attempt to solve the MINLP problems we implemented the FFS approach described in 7.9 with two different problems that are available at [16] under the name st_e35 and gasnet. The tests were conducted considering that all four versions of the heuristic algorithm will be running for five replications, here the number of iterations

depend on the stopping criteria which if we recall from pseudocode 7.9 is when $\sum_{k=1}^{K} \delta_k \leq 10^{-5}$. Results presented in Table 7.1 reflect the behaviour of the FSS approach. The first row of Table 7.1 shows the name of the instances considered, the next five rows give general information about each instance followed by the best-known solution and the solver used, all of this taken from [16]. Below there are four blocks of results recorded from the FSS implementation and the three variants explained above. We present the value achieved for the objective function, in order to have an insight we present the proportion of feasible solutions (number of times that we obtained a feasible solution over the total number of solutions), the proportion of optimal solutions over those being feasible (number of optimal solutions/number of feasible solutions) and finally we report the total computational time spent.

Results from Table 7.1 indicate that for "st_e35" problem the FSS implementation along with the other three tested variants reached the best-known solution. In terms of computation time "st_e35" problem can be solved within a range of 72.27 to 90.27 seconds. By contrast, results in Table 7.1 for "gasnet" problem indicate that our best solution 6999400.00 was obtained with the original FSS approach in 11.24 hours. This contrasting computational time spent between the two problems may be explained by their own nature, "st_e35" problem consist of linear constraints and a non-linear objective function, however "gasnet" problem consist of a linear objective function with linear and non-linear constraints, which makes it more difficult to find a feasible solution. Considering that this implementation of our FSS approach did not reach the best solution for both instances tested (solely for the problem with only linear constraints) we did not pursue it into more instances, instead we decided to rethink our approach and to change of non-linear solver. The work carried out is fully described in section 7.4.

Table 7.1: Results from FSS implementation

Problem name	st_e35	gasnet
objective function	non-linear	linear
No of linear constraints	39	25
No of non-linear constraints	0	44
Total no of variables	32	90
No of binary variables	7	10
Best-known solution (from [16])	64868.08	6999381.56
Solver used (from [16])	Baron	Baron
Results from adding $\Sigma_{i\in P_k} x_i(1-x_i) \leq \delta_k \;\; k = 1, ..., K$		
objective function value	64868.0768	6999400.00
(feasible/total) snopt solutions	0.88	0.10
(optimal/feasible) snopt solutions	0.07	0.20
Total time in seconds	72.36	40472 (11.24 hrs)
Results from considering $\min c(x,y) + \mu\Sigma_i x_i(1-x_i)$		
Objective function value	64868.0768	7004600.00
(feasible/total) snopt solutions	0.88	0.07
(optimal/feasible) snopt solutions	0.07	0.33
time in seconds	90.27	29188 (8.11 hrs)
Results from adding $\Sigma_{i\in P_k} \lvert x_i \rvert \lvert (1-x_i) \rvert \leq \delta_k \;\; k = 1, ..., K$		
objective function value	64868.0768	7007449.96
(feasible/total) snopt solutions	0.88	0.11
(optimal/feasible) snopt solutions	0.21	0.14
time in seconds	72.27	45541.62 (12.65hrs)
Results from considering $\min c(x,y) + \mu\Sigma_i \lvert x_i \rvert \lvert (1-x_i) \rvert$		
objective function value	64868.0768	7004607.81
(feasible/total) snopt solutions	1.00	0.14
(optimal/feasible) snopt solutions	0.06	0.02
time in seconds	85.08	8 hrs

7.4 Second approach FSS-MINLP-2

As a consequence of the results obtained in 7.3.3 we continued our research and in this section we present a new adaptation of the FSS approach proposed in 7.3.1 to solve mixed-integer non-linear problems. We present computational results using a recently developed solver called Minotaur [46].

Let us recall here that an MINLP problem is given by equations (7.1)-(7.4), in order to motivate our FSS approach first observe that the condition x is zero-one (equation (7.3)) can be replaced by equations (7.5) and (7.6)

Here we have replaced the explicit integrality condition on x given in equation (7.3) by the condition that x is continuous (between zero and one, equation (7.6)), but with integrality being enforced by the single non-linear equality constraint, equation (7.5).

Now given the capabilities of non-linear optimisation software one might suspect that simply replacing an explicit integrality condition by equations (7.5) and (7.6) would not be computationally successful, since we would be hoping to generate a (globally optimal) solution to a continuous non-linear optimisation problem with a very tight equality constraint (equation (7.5)). Limited computational experience with the solver we used, Minotaur [46], in fact indicated that even generating feasible solutions when equation (7.5) was present in the formulation was difficult.

Hence, drawing on our experience with FSS, we replace equation (7.5) by:

$$\sum_{i=1}^{n} x_i(1 - x_i) \quad \leq \delta \tag{7.18}$$

Conceptually by initially setting δ to a suitably high value, then reducing it in systematic fashion, we are solving a sequence of different formulations of the problem. In this process we hope to be able to generate good-quality feasible solutions. Our algorithm for this is given in section 7.4.1.

7.4.1 Algorithm

The pseudocode for our FSS algorithm for MINLP's is presented in Algorithm 7.10. We start by removing the integrality requirement and solving the continuous relaxation of the problem (equations (7.1)-(7.2),(7.4),(7.6)). Let the solution for the relaxed zero-one variables be X_i $i = 1, \ldots, n$. We can then set an initial value for δ using $\delta = \sum_{i=1}^{n} X_i(1 - X_i)$.

We now iterate, solving equations (7.1)-(7.4),(7.18) and reducing δ by a factor α $(0 < \alpha < 1)$ at each iteration. Note that the optimisation problem we solve here is the original MINLP (equations (7.1)-(7.4)), with the addition of equation (7.18). The idea is that adding this constraint perturbs the formulation and hence, given the nature of any non-linear solution software, may well lead to a different solution. We terminate when δ is small $(\leq 10^{-5})$ or we have performed a number of consecutive iterations (three iterations) without changing the value of the best solution found. Based on limited computational experience we reduced δ by a factor $\alpha = 0.9$ at each iteration. In the pseudocode seen in Algorithm 7.10 z_{best} is the best feasible solution found.

7.4.2 Computational results

7.4.2.1 Test problems

The results from our approach were obtained using an Intel(R) Core(TM) 3.3 GHz CPU with 4.0 GB. The algorithm was implemented in AMPL [69] using the Minotaur solver [46]. Minotaur is a recently developed toolkit for solving mixed-integer non-linear optimisation problems. It has two main solvers, one based on non-linear branch-and-bound denoted as bnb, the other an implementation of a QP-diving algorithm. Here we used the bnb solver.

We considered 51 standard benchmark problems taken from MINLPLib [12, 16]. Table 7.2 shows the problem considered. Note that many of these problems are very

Algorithm 7.10 FSS-MINLP-2 for mixed-integer non-linear problems pseudocode

Initialisation: $\alpha \leftarrow 0.9 \;\; z_{best} \leftarrow \infty \;\; t \leftarrow 0 \;\; t_z \leftarrow 0$

$[X_i] \leftarrow$ solve equations $(7.1) - (7.2), (7.4), (7.6))$ {Solve the continuous relaxation}

Set $\delta \leftarrow \sum_{i=1}^{n} X_i(1 - X_i)$

Iterative process:

repeat

$t \leftarrow t + 1$ {Update the iteration counter}

$z \leftarrow$ solve equations $(7.1) - (7.4), (7.18)$

$z_{best} \leftarrow \min[z, z_{best}]$ {Update the best solution found}

if $z_{best} = z_{best}$ at iteration $(t - 1)$ **then**

$t_z \leftarrow t_z + 1$ {Update counter t_z in iteration t}

end if

update $\delta \leftarrow \alpha\delta$

until $\delta \leq 10^{-5}$ or $t_z = 3$

large. On average these problems have 1073 constraints and 721 continuous, 566 zero-one, variables each.

With regard to whether (or not) these MINLP's are convex MINLPlib does not categorise problems as to their convexity status [77]. Hence we have attempted a categorisation based on information given by other authors (Tables 8.1 and 8.2 in [15], Table 6 in [47]) and this is shown in the last column of Table 7.2. In that column n/k indicates not known, unclear indicates that the sources cited differ as to their classification of a problem. As is stressed in [15] this convexity classification should not be regarded as completely accurate, however the information shown with regard to convexity in Table 7.2 is the best that we could obtain.

With reference to our choice of solver note here that results given recently in [61] indicate that Minotaur is currently one of the best solvers for the MINLP's in MINLPLib.

Table 7.2: MINLPLib problems considered

Name	Number of			Convex?
	constraints	continuous variables	zero-one variables	
contvar	285	209	88	n/k
csched1	23	14	63	no
csched2	138	93	308	unclear
csched2a	138	93	140	no
deb10	130	161	22	no
deb6	508	456	20	no
deb7	898	794	20	no
deb8	898	804	20	no
deb9	918	794	20	no
eniplac	190	118	24	no
enpro48pb	215	62	92	no
enpro56pb	192	55	73	no
ex1265	75	31	100	no
ex1266	96	43	138	no
gasnet	70	81	10	no
minlphix	93	65	20	n/k
netmod_dol1	3138	1537	462	n/k
netmod_dol2	3081	1537	462	n/k
nuclear10a	3340	2091	10920	no
nuclear14	1227	987	576	no
nuclear14a	634	393	600	no
nuclear14b	1786	969	600	no
nuclear24	1227	987	576	no
nuclear24a	634	393	600	no
nuclear24b	1786	969	600	no
nuclear25	1304	1054	625	no
nuclear25a	660	409	650	no
nuclear25b	1910	1034	650	no
nuclear49	3874	3335	2401	no
nuclear49a	1432	892	2450	no
nuclear49b	6234	3293	2450	no
nuclearvf	318	184	168	no
ortez	75	70	18	no
parallel	116	181	25	no
ravempb	187	59	54	no
risk2b	581	450	14	n/k
risk2bpb	581	450	14	yes
saa_2	6206	4008	400	no
space25	236	144	750	no
spectra2	73	40	30	no
stockcycle	98	49	432	yes
super1	1659	1264	44	no
super2	1659	1264	44	no
super3	1659	1264	44	no
super3t	1343	1012	44	no
synheat	65	45	12	no
util	168	118	28	no
waste	1992	2085	400	no
water4	138	70	126	no
waterful2	384	182	448	n/k
waterx	55	57	14	no

7.4.3 FSS versus Minotaur

Our computational results for FSS and Minotaur are given in Table 7.3. In the first column we give the name of the problem and in the second column the value of the best-known solution as taken directly from [16]. Columns three and four give the solution obtained by Minotaur when it solves the original MINLP together with the corresponding computation time (in seconds). Here if the solution is equal to the best-known solution we indicate this by the word "best". Note that for some of these problems Minotaur is unable to generate a feasible solution. Columns five and six give the solution obtained by our FSS approach, together with the corresponding solution time. Column seven compares the FSS solution value with that obtained by Minotaur. With regard to time limits we imposed a time limit of 18000 seconds (5 hours) for both Minotaur and FSS and problems that terminated at time limit are shown by the time being bracketed in Table 7.3.

Considering Table 7.3 and the last column comparing the FSS solution with the Minotaur solution we have that:

- for 30 of the 51 problems considered the two approaches give the same solution, with the average time for FSS being 1665.9 seconds, as compared with 1731.6 seconds for Minotaur. Hence for these problems we can conclude that FSS is computationally competitive with Minotaur, giving the same result in approximately the same (average) time.

- for 10 of the 51 problems considered (namely csched2a, enpro56pb, ex1265, gasnet, nuclearvf, risk2b, risk2bpb, water4, waterful2 and waterx) FSS gives a better quality solution than Minotaur. Here the average times are 459.7 seconds for FSS, but 1529.0 seconds for Minotaur. In other words FSS gives ten better quality solutions in (on average) 30% of the time taken by Minotaur. Looking at the average percentage solution gap (as measured by averaging 100(solution value - best-known

Table 7.3: FSS results

Name	Best-known solution	Minotaur		FSS		FSS solution versus Minotaur solution
		Solution	Time (s)	Solution	Time (s)	
contvar	809149.8	best	9.7	best	59.8	equal
csched1	-30639.3	best	0.1	best	0.7	equal
csched2	-166102	infeasible	0.1	infeasible	0.1	equal
csched2a	-165399	-163906	44.5	best	224.6	better
deb10	209.43	best	0.2	best	1.0	equal
deb6	201.7393	229.0111	3.3	229.0111	3.7	equal
deb7	116.5846	168.9588	29.7	infeasible	0.9	worse
deb8	116.5846	168.9588	38.0	168.9588	47.1	equal
deb9	116.5846	168.9588	23.4	infeasible	0.7	worse
eniplac	-132117	best	193.8	best	718.6	equal
enpro48pb	187277.3	best	2.6	best	11.5	equal
enpro56pb	263428.3	infeasible	0.1	best	19.7	better
ex1265	10.3	15.1	0.1	best	261.3	better
ex1266	16.3	best	1.4	best	6.1	equal
gasnet	6999382	7004608	0.3	best	0.7	better
minlphix	316.6927	best	0.1	best	0.1	equal
netmod_dol1	-0.56	best	(18000)	infeasible	5.4	worse
netmod_dol2	-0.56	best	511.8	best	17352.0	equal
nuclear10a	-	infeasible	(18000)	infeasible	(18000)	equal
nuclear14	-1.1277	-1.1286	12.4	-1.1286	12.3	equal
nuclear14a	-1.1296	-1.1292	5.7	-1.1292	5.8	equal
nuclear14b	-1.1169	-1.123	10203.5	-1.121	14405.1	worse
nuclear24	-1.1277	-1.1286	12.3	-1.1286	12.2	equal
nuclear24a	-1.1296	-1.1292	5.6	-1.1292	5.8	equal
nuclear24b	-1.1169	-1.12301	(18000)	-1.121	14403.4	worse
nuclear25	-1.1186	-1.11983	540.0	-1.11937	539.8	worse
nuclear25a	-1.1202	-1.12001	3731.4	-1.12001	3787.4	equal
nuclear25b	-1.101	-1.10807	(18000)	-1.1062	10802.8	worse
nuclear49	-1.1514	-1.1513	10085.7	-1.1513	5193.1	equal
nuclear49a	-1.1514	best	1225.1	best	198.4	equal
nuclear49b	-1.1169	infeasible	(18000)	infeasible	3609.9	equal
nuclearvf	-1.0225	-1.01612	2.7	-1.01944	23.5	better
ortez	-9532.04	best	0.1	best	0.2	equal
parallel	924.2956	best	0.2	best	1.3	equal
ravempb	269590.2	best	0.3	best	1.8	equal
risk2b	-56.8208	-55.8761	1.0	best	6.3	better
risk2bpb	-56.8208	-55.8761	1.0	best	6.3	better
saa_2	12.1613	infeasible	10.7	infeasible	8.3	equal
space25	484.3286	best	27.2	infeasible	0.4	worse
spectra2	13.9783	best	0.7	best	4.0	equal
stockcycle	119948.7	best	56.5	best	919.2	equal
super1	12.6284	infeasible	3.5	infeasible	3.4	equal
super2	9.5079	infeasible	28.2	infeasible	7.0	equal
super3	4.9345	infeasible	7.5	infeasible	5.0	equal
super3t	-0.6689	-0.6825	8141.8	infeasible	0.3	worse
synheat	154997.3	189521.1	0.1	infeasible	0.1	worse
util	999.5788	999.5866	0.3	infeasible	0.1	worse
waste	598.9192	infeasible	0.1	infeasible	0.1	equal
water4	907.017	910.6469	89.5	best	451.6	better
waterful2	1012.609	980.9673	15148.4	968.1149	3602.3	better
waterx	909.0401	934.8622	1.9	934.8594	0.2	better

solution)/(| best-known solution |) over the ten problems) this is -0.13% for FSS but 5.74% for Minotaur. Here the negative sign for the FSS gap indicates that (on average) FSS improved on the best-known solution values given in [16]. In calculating this average percentage solution gap we ignore the single problem (enpro56pb) where Minotaur did not find a feasible solution. Hence for these problems we can conclude that FSS is far superior to Minotaur, achieving significantly higher quality solutions in much less time.

- for 11 of the 51 problems considered (namely deb7, deb9, netmod_dol1, nuclear14b, nuclear24b, nuclear25, nuclear25b, space25, super3t, synheat and util) FSS gives a worse solution than Minotaur. Here the average times are 3650.8 seconds for FSS, but 6633.3 seconds for Minotaur. Within these 11 problems there are four problems (nuclear14b, nuclear24b, nuclear25 and nuclear25b) where both FSS and Minotaur find a feasible solution. For these four problems the average percentage solution gap is -0.32% for FSS and -0.46% for Minotaur, with the average times being 10037.8 and 11685.9 seconds for FSS and Minotaur respectively. For all four of these problems both FSS and Minotaur improved on the best-known solution given in [16]. For the remaining seven problems FSS did not find a feasible solution (average time 1.1 seconds), whilst Minotaur did find feasible solutions, the average percentage solution gap being 15.73% in an average time of 3746.1 seconds).

7.4.4 FSS versus minlp_bb and RECIPE

Table 7.3 has compared our FSS approach against applying a single MINLP solver (Minotaur) directly. In order to see how our FSS approach compares against other approaches in the literature for finding good MINLP solutions we can compare our FSS results against those recently published by Liberti et al [47] who presented results for a Relaxed-Exact Continuous-Integer Problem Exploration algorithm (RECIPE). RECIPE combines a global search phase based on variable neighbourhood search (using two sep-

arate neighbourhoods) and a local search phase based on a MINLP heuristic. They also presented results for solving MINLP's directly using minlp_bb, a MINLP solver which Liberti et al [47] found to be extremely fast. See also Liberti et al [44].

Table 7.4 shows the comparison. In that table we show the FSS results (as in Table 7.3, but repeated here for ease of comparison) and the minlp_bb and RECIPE results. These minlp_bb and RECIPE results have been taken directly from Table 1 of [47], but have been adjusted to give the benefit of the doubt to minlp_bb and RECIPE where the solution value given in [47] is numerically very close (but not identical) to the best-known solution value. Table 7.4 contains 50 problems, one less than Table 7.3 (waterful2 not being solved in [47] for technical reasons). The computation times given for minlp_bb and RECIPE are in seconds on an Intel Xeon X3353 2.66 GHz with 24 GB RAM running Linux.

In Table 7.4 there are two sets of columns for RECIPE, one where Ipopt is used as the nonlinear solver, the other where filterSQP is used as the nonlinear solver (these are continuous non-linear solvers, in both cases minlp_bb is used as the MINLP solver). Computationally a time limit of two hours (7600 seconds) was applied to the results for minlp_bb and RECIPE shown in Table 7.4.

Table 7.5 compares FSS with the various algorithms in Table 7.4, as well as with the results from Table 7.3. In that table we give the number of problems where the solution from FSS and that of another algorithm are better, equal or worse. In terms of computation time the average time for FSS for the 50 problems in Table 7.4 was 1822.6 seconds, albeit on a different pc from the one used to produce the results for Liberti et al [47].

In terms of quality of solution it seems clear from Table 7.5 that FSS produces better solutions than minlp_bb or RECIPE/filterSQP. Over the 50 problems considered minlp_bb only produces a better solution than FSS 7 times, RECIPE/filterSQP only produces a better solution than FSS 9 times.

Table 7.4: Liberti et al [47] results

Name	Best-known	FSS		minlp_bb		RECIPE/Ipopt		RECIPE/filterSQP	
	solution	Solution	Time (s)	Solution	Time (s)	Solution	Time (s)	Solution	Time (s)
contvar	809149.8	best	59.8	best	115.62	best	7201.72	infeasible	99.69
csched1	-30639.3	best	0.7	best	0.64	best	156.50	best	1.73
csched2	-166102	infeasible	0.1	infeasible	0.07	best	7315.06	infeasible	18.06
csched2a	-165399	best	224.6	infeasible	0.01	best	3286.04	infeasible	72.76
deb10	209.43	best	1.0	infeasible	0.10	infeasible	674.73	infeasible	4.97
deb6	201.7393	229.0111	3.7	best	12.98	infeasible	1217.27	infeasible	3.41
deb7	116.5846	infeasible	0.9	infeasible	121.30	infeasible	4544.21	infeasible	10.89
deb8	116.5846	168.9588	47.1	infeasible	91.30	infeasible	4482.77	infeasible	17.06
deb9	116.5846	infeasible	0.7	infeasible	119.40	infeasible	4920.32	infeasible	11.05
eniplac	-132117	best	718.6	-131863.6349	25.27	best	7200.05	best	477.89
enpro48pb	187277.3	best	11.5	best	2.57	best	560.52	best	14.08
enpro56pb	263428.3	best	19.7	best	20.36	best	2515.05	best	49.50
ex1265	10.3	best	261.3	15.1	0.04	15.1	8.92	best	6.40
ex1266	16.3	best	6.1	best	0.16	best	19.24	best	7.52
gasnet	6999382	best	0.7	7004607.8064	6.60	6999391.6436	117.70	7045336.9264	153.55
minlphix	316.6927	best	0.1	infeasible	0.00	best	54.76	best	3.38
netmod_dol1	-0.56	infeasible	5.4	infeasible	9698.77	infeasible	9720.59	infeasible	9716.51
netmod_dol2	-0.56	best	17352.0	best	2470.45	-0.5535	8930.19	best	9008.98
nuclear10a	—	infeasible	(18000)	infeasible	0.00	infeasible	0.00	infeasible	0.00
nuclear14	-1.1277	-1.1286	12.3	infeasible	41.10	-1.1257	6062.20	infeasible	1378.68
nuclear14a	-1.1296	-1.1292	5.8	-1.128	160.73	best	2732.95	-1.128	1270.87
nuclear14b	-1.1169	-1.121	14405.1	-1.0896	7221.04	-1.1093	7200.37	-1.0936	7200.60
nuclear24	-1.1277	-1.1286	12.2	infeasible	41.04	-1.1257	6006.97	infeasible	1383.67
nuclear24a	-1.1296	-1.1292	5.8	-1.128	160.40	best	2769.73	-1.128	1272.52
nuclear24b	-1.1169	-1.121	14403.4	-1.0896	7212.66	-1.1093	7201.06	-1.0936	7205.14
nuclear25	-1.1186	-1.11937	539.8	infeasible	64.80	-1.1171	7225.31	infeasible	1840.91
nuclear25a	-1.1202	-1.12001	3787.4	-1.1193	622.00	infeasible	0.00	-1.1	7217.52
nuclear25b	-1.101	-1.1062	10802.8	-1.0851	7200.35	-1.0977	7201.25	infeasible	7202.16
nuclear49	-1.1514	-1.1513	5193.1	infeasible	865.66	infeasible	7200.83	infeasible	7298.06
nuclear49a	-1.1514	best	198.4	best	4185.18	infeasible	8293.68	infeasible	8188.66
nuclear49b	-1.1169	infeasible	3609.9	infeasible	0.00	infeasible	0.00	infeasible	0.00
nuclearvf	-1.0225	-1.01944	23.5	infeasible	2.52	best	1911.18	infeasible	212.76
ortez	-9532.04	best	0.2	best	0.07	best	68.70	best	0.77
parallel	924.2956	best	1.3	best	0.59	best	1.14	best	3.31
ravempb	269590.2	best	1.8	best	0.43	best	57.39	best	19.68
risk2b	-56.8208	best	6.3	infeasible	0.02	best	305.67	infeasible	4.20
risk2bpb	-56.8208	best	6.3	-55.8761	3.36	best	1631.20	best	25.52
saa_2	12.1613	infeasible	8.3	infeasible	67.32	infeasible	0.00	infeasible	7230.22
space25	484.3286	infeasible	0.4	best	72.29	best	1146.59	best	223.88
spectra2	13.9783	best	4.0	best	0.12	best	228.87	best	5.90
stockcycle	119948.7	best	919.2	best	31.34	best	5124.47	best	3515.12
super1	12.6284	infeasible	3.4	infeasible	51.04	9.6438	7224.29	9.8913	7207.42
super2	9.5079	infeasible	7.0	infeasible	33.80	5.2468	7221.22	5.2907	7210.68
super3	4.9345	infeasible	5.0	infeasible	45.77	12.9385	7205.99	13.4772	7210.18
super3t	-0.6689	infeasible	0.3	-0.6744	7206.93	-0.6684	7206.03	-0.6673	7201.32
synheat	154997.3	infeasible	0.1	best	0.09	best	11.13	best	1.56
util	999.5788	infeasible	0.1	best	2.00	best	313.65	best	4.31
waste	598.9192	infeasible	0.1	732.2534	7200.18	679.0943	7208.15	903.8701	7200.66
water4	907.017	best	451.6	best	66.48	best	298.20	best	63.23
waterx	909.0401	934.8594	0.2	926.5789	3.27	914.1837	142.69	919.051	9.07

In terms of a comparison between FSS and RECIPE/Ipopt the situation is less clear-cut. Looking in detail at those results:

- of the 15 problems where FSS give a better solution: for six problems FSS produces a feasible solution whereas RECIPE/Ipopt does not, for the other nine problems the average percentage solution gap is -0.16% for FSS but 5.55% for RECIPE/Ipopt.
- of the 13 problems where RECIPE/Ipopt give a better solution: for nine problems RECIPE/Ipopt produces a feasible solution whereas FSS does not, for the other four problems the average percentage solution gap is 0.80% for FSS but 0.14% for RECIPE/Ipopt.

We would note here however that, as Table 7.5 shows, RECIPE/Ipopt is the computationally most expensive algorithm of those given in Liberti et al [47].

7.4.4.1 Discussion

Table 7.5 indicates that our FSS approach has a role to play within MINLP. As can be seen there it is competitive, in terms of quality of solution, with other approaches: either stand-alone general MINLP solvers such as Minotaur or minlp_bb, or more specialised algorithms such as RECIPE/Ipopt and RECIPE/filterSQP.

In more detail Table 7.5 indicates that (in terms of solution quality) FSS is (on balance) superior to both minlp_bb and RECIPE/filterSQP, roughly equivalent to both Minotaur and RECIPE/Ipopt. In terms of computation time the different hardware used makes a direct comparison difficult, but the average times quoted in Table 7.5 do not seem to significantly favour one approach over any other.

7.5 Conclusions

In this chapter we have addressed two approaches based on formulation space search to solve mixed-integer non-linear (zero-one) programming problems.

Table 7.5: FSS comparison

	FSS versus Minotaur	FSS versus minlp_bb	FSS versus RECIPE/Ipopt	FSS versus RECIPE/filterSQP
Number of better solutions using FSS	10	20	15	19
Number of equal solutions using FSS	30	23	22	22
Number of worse solutions using FSS	11	7	13	9
Average FSS time (seconds, Intel 3.3 GHz)	1857.5			
Average Minotaur time (seconds, Intel 3.3 GHz)	2749.1			
Average FSS time (seconds, Intel 3.3 GHz)		1822.6	1822.6	1822.6
Average minlp_bb time (seconds, Intel Xeon 2.66 GHz)		1105.0		
Average RECIPE/Ipopt time (seconds, Intel Xeon 2.66 GHz)			3442.5	
Average RECIPE/filterSQP time (seconds, Intel Xeon 2.66 GHz)				2369.7

With regards to our first approach (FSS-MINLP-1), limited computational experience suggested that we needed to consider a different way to tackle mixed-integer non-linear problems using formulation space search and certainly a different non-linear solver.

Our second approach (FSS-MINLP-2) is an iterative process based on adding a single non-linear inequality constraint of increasing tightness to the original problem. Computational results were presented for 51 standard benchmark problems taken from MINLPLib.

We compared our formulation space search approach (FSS-MINLP-2) against the Minotaur non-linear solver, as well as against the minlp_bb non-linear solver and the recently published RECIPE algorithms.

Overall, in terms of quality of solution, our approach was superior to minlp_bb and one of the RECIPE algorithms (RECIPE/filterSQP), competitive with respect to Minotaur and another RECIPE algorithm (RECIPE/Ipopt).

Formulation space search is a new and emerging metaheuristic, especially applicable to non-linear optimisation problems, and we believe that the results presented here with FSS-MINLP-2 indicate that it provides a useful addition to the armoury of algorithms available to solve mixed-integer non-linear (zero-one) programming problems.

Chapter 8

Conclusions, contributions and future work

This final chapter presents a summary of the implementation of the formulation space search method for the circle packing problem with identical circles and the reformulation descent method for the circle packing problem with non-identical circles considering different variants in both cases. As well we discuss the extension of the formulation space search algorithm to the mixed integer non-linear integer programming problem, in particular we focused on those problems where integer variables are defined as zero-one variables (binary variables).

8.1 Conclusions

This dissertation has investigated the scope of the formulation space search and reformulation descent methods in particular when implemented through a heuristic algorithm to solve circle packing problems.

We have presented a general overview of the most relevant literature (to our approach) concerning the packing problem addressing the case for identical circles and non-identical circles, and considering different shaped containers such as: circular, square, rectangular

and triangular containers. We as well presented a brief historical note on the packing problem highlighting applications. We also presented the literature related to formulation space search and its applications.

We have proposed and implemented a heuristic algorithm based on the formulation space search method for the packing problem with identical circles, the aim of this approach consisted of maximising the radius of the circles to be packed inside a container of fixed size. We successfully implemented our heuristic and considered seven different instances that depend on the shape of the container chosen: the unit circle, the unit square, a rectangle with dimensions $(L = 5, W = 1)$, a rectangle with dimensions $(L = 10, W = 1)$, a right-angled isosceles triangle with length $(L = 1)$, a semicircle with radius 1 and a right-circular quadrant with radius 1.

For the case of non-identical circles we investigated an approach using the formulation space search aiming to find the minimum size of the circular container where the circles to be packed have fixed size. For this approach in order to guarantee a feasible solution we adopted a correction step based on a greedy strategy that for small number of circles achieved and even improved the best-known solution however, when increasing the size of the problem there was enough evidence suggesting that we should rethink the approach. Hence, we moved on and implemented the packing problem with non-identical circles with a new scaled formulation adopting a reformulation descent approach, this time we were able to consider the seven instances that depend on the shaped container that we described above. Although in the literature we found some packing problems considering rectangular containers, the problems being addressed (minimise the perimeter of the rectangle, regard one dimension of the rectangle as fixed and minimise the other dimension) are not equivalent to the scaled problem we presented in this thesis, hence we considered a new problem.

In order to explore the capabilities of the formulation space search we decided to implement it for mixed-integer non-linear programming problems, in particular we fo-

cused on problems with integer variables defined as zero-one variables (binary). The results obtained from the first implementation suggest that we have not chosen an adequate solver for our purpose, for one of the problems the results achieved the optimal solution in a fast total time, while for the other problem which by nature constitutes a harder problem to solve (having linear and non-linear constraints) we only achieve a feasible solution close, but subjectively not enough, to the best-known solution. Regarding our second implementation, this was tested using a set benchmark problems, the results obtained suggest that using this approach provides a competitive way to solve mixed-integer non-linear programming problems when compared with other approaches.

8.2 Contributions

We have given a competitive heuristic algorithm for the packing problem for the case of identical circles when balancing accuracy of solution and computational time spent. We also were able to produce improvements over the previous best-known solutions for containers such as the semicircle, the right-angled triangle and the right-circular quadrant. All this work is presented in the paper [54] that we published in the European Journal of Operational Research in 2011.

We provided a new approach for the solution of the packing problem with non-identical circles based on a scaled formulation model using reformulation descent. Although the scaled formulation of the packing problem is an equivalent formulation of the approach adopted before with a greedy strategy, results suggest that in practice it is more likely to obtain better quality solutions with this scaled formulation. The work carried out is presented in the paper [55] that was published in the Computers & Operations Research journal early in 2013.

The instances considered are classified as large variation instances and small variation instances. Here we gave results for the unit circle container and presented new results for another six different containers for both large and small variation instances. Although for

the case of non-identical circles it was not possible to generalize a strategy to reduce the size of the non-overlapping constraints, we considered dropping the smaller circles for the optimisation process only for the large variation instances, and after the improvement process we adopted an efficient way (the insertion process) to locate the remaining circles.

According to the solutions available from our implementation of FSS to mixed-integer non-linear zero-one problems, based on a couple of instances, suggest that solely using a non-linear solver such as SNOPT [18, 35] may not be enough to give a solution to such problems. Although our findings are limited they indicate that when the problem at hand has only linear constraints and non-linear objective function SNOPT is able to produce the optimal solution, however a different scenario is presented when we consider more difficult problems. In the light of this information we believe that adopting a more suitable solver may lead us to better results when implementing our FFS approach. Hence, we decided to implement a modified version of the FSS approach for mixed-integer non-linear zero-one problems using a different solver called Minotaur [46]. Our findings (based on 51 benchmark problems taken from MINLPLib [16]) showed that this FSS implementation was competitive when compared with other algorithms. This work can be found in the paper [53] that was recently accepted in the Optimization Letters journal early in 2013.

8.3 Future Work

This research has thrown up many paths to follow relating to the circle packing problem and as well addressing mixed-integer non-linear programming problems.

The 3-Dimensional packing problem. As an extension of the work done for the 2-dimensional case of the packing problem we could apply our heuristic to the 3-dimensional case which is as well known as the sphere packing problem. The problem concerns with arrangement of a certain number n of identical spheres inside the unit sphere aiming to find the maximum radius for the small spheres satisfying the non-

overlapping conditions between the spheres and the container.

In the same vein we could consider as well the implementation of our FSS heuristic algorithm to the packing problem with identical spheres in different containers, having as basis those considered in Chapter 4 in three dimensions such as: a unit box, several rectangular boxes with length (L), width(W) and depth (D) determined as $(L = 5, W = 1, D = 1)$ or $(L = 5, W = 1, D = 5)$ or $(L = 10, W = 1, D = 1)$ or $(L = 10, W = 1, D = 10)$ just to mention some examples related with the original 2-dimensional case of rectangle $(L = 5, W = 1)$ and rectangles $(L = 10, W = 1)$, in a similar way we could consider some variants of the right-angle isosceles triangle with length $L = 1$, the semicircular and the right-quadrant circular container with radius one, having depth greater than zero.

Another interesting avenue for future work would be implementing the scaled formulation for non-identical circles presented in Chapter 5 to the 3-dimensional version of packing spheres with non-identical size, from here an obvious extension would be considering different containers as those given in Chapter 6 this time in 3-dimensions.

Packings of variable-sized containers with maximum total sum of perimeters or areas. This packing problem has an interesting point of view, it is available in Packomania web site [72]. The problem involves a fixed size rectangular container aiming to determine the size of each non-identical circle in order to maximize the total area covered by them, or to maximise the sum of the perimeters of the circles packed. As the perimeter is a function of the radius we can see this as the maximum of the sum of the radii of the circles. Using our FSS approach we could addresses this problem and also consider extending the problem to different containers.

The circle packing problem with forbidden areas. Extending the FSS method to the packing problem with forbidden areas. This problem was recently presented in [75], the aim of this problem is to maximize the number of identical circles inside a container with fixed circular and/or union of circular prohibited areas.

Appendix A

An alternative approach for the case of non-identical circles

Our first attempt to solve the packing problem with non-identical circles aiming to minimise the size of the circular container for large variation instances it is generally based on the reformulation descent approach. However, in order to minimise the size of the container we put special attention to the correction step by implementing a strategy based on the general procedure of a greedy algorithm. In what follows we present the formulation used, the heuristic, and the description of the correction step that follows a greedy approach. We also present computational results, although this approach was less successful than the scaled formulation given in Chapter 5.

A.1 Formulation and heuristic

Based on the reformulation descent we give the mixed formulation in equations (A.1)-(A.11). The objective function (A.1) minimises the size of the circular container, equation (A.2) ensure that the circles are fully inside the container for those expressed in Cartesian coordinates whilst equation (A.3) is the polar equivalent. Equation (A.4) prevent overlaps for any two circles, hence they are known as the non-overlapping con-

straints. Equations (A.5)-(A.6) are as equation (A.4) but they consider that one or both centres are expressed in polar coordinates. Equations (A.7)-(A.11) are the limits on the variables. Here $R_{overlap} = \sqrt{\sum_{i=1}^{n} R_i^2}$ is the lower bound on the variable R, it is deduced by area considerations ($\pi R_{overlap}^2 = \sum_{i=1}^{n} \pi R_i^2$).

$$\min \quad R \tag{A.1}$$

st

$$x_i^2 + y_i^2 \leq (R - R_i)^2 \qquad \forall i \in C \tag{A.2}$$

$$r_i \leq R - R_i \qquad \forall i \in P \tag{A.3}$$

$$(x_i - x_j)^2 + (y_i - y_j)^2 \geq (R_i + R_j)^2 \qquad \forall (i,j) \in Q \text{ with } i,j \in C \; i < j \tag{A.4}$$

$$(x_i - r_j \cos(\theta_j))^2 + (y_i - r_j \sin(\theta_j))^2 \geq (R_i + R_j)^2 \quad \forall (i,j) \in Q \text{ with } i \in C \; j \in P \tag{A.5}$$

$$r_i^2 + r_j^2 - 2 r_i r_j \cos(\theta_i - \theta_j) \geq (R_i + R_j)^2 \qquad \forall (i,j) \in Q \text{ with } i,j \in P \; i < j \tag{A.6}$$

$$-\sum_{i=1}^{n} R_i \leq x_i \leq \sum_{i=1}^{n} R_i \qquad \forall i \in C \tag{A.7}$$

$$-\sum_{i=1}^{n} R_i \leq y_i \leq \sum_{i=1}^{n} R_i \qquad \forall i \in C \tag{A.8}$$

$$0 \leq r_i \leq \sum_{i=1}^{n} R_i \qquad \forall i \in P \tag{A.9}$$

$$0 \leq \theta_i \leq 2\pi \qquad \forall i \in P \tag{A.10}$$

$$R_{overlap} \leq R \tag{A.11}$$

In pseudocode A.11 we give an overview of the heuristic algorithm implemented using the reformulation descent approach. This implementation is essentially the same as Algorithm 3.5 for the case of identical circle: in the initialisation step we set the R_{best} as ∞ in order to keep the minimum size of the container at every iteration, we generate

Algorithm A.11 Reformulation descent pseudocode for the case of non-identical circles

Function $(R_{best}, X_{best}, Y_{best}) \leftarrow FSS(n, replication\_limit, iteration\_limit)$

Initialisation: $|C| \leftarrow \lfloor n/2 \rfloor$ $R_{best} \leftarrow \infty$ $t\_rep \leftarrow 0$

repeat

$t \leftarrow 0$

$(x^0, y^0) \leftarrow InitialSolution(n, |C|)$ {a first initial solution}

repeat

$Q \leftarrow OverlapSet(n)$ {give the overlap set Q}

$(x, y, R) \leftarrow NLP(x^0, y^0, C, P, Q, R_{overlap})$

$(R^*, x, y) \leftarrow Correction(x^0, y^0, x, y)$ {correct the radius}

$R_{best} \leftarrow \max\{R_{best}, R^*\}$ {update R_{best}}

$(X_{best}, Y_{best}) \leftarrow (x, y)$ {save coordinates associated with R_{best}}

$t \leftarrow t + 1$ {update iteration counter}

$(x^0, y^0) \leftarrow (x, y)$ {sets initial solution}

$C \leftarrow P$ $P \leftarrow \{1, ..., n\} \setminus C$ {switch the sets C and P}

until $t = iteration\_limit$

$t\_rep \leftarrow t\_rep + 1$ {update replication counter}

until $t\_rep = replication\_limit$

a random initial solution. In step two we generate $n(n-1)/2$ different pairs of circles that will be elements of the Q set (the set with pair of circles that may overlap). In step three we obtain a solution from the non-linear solver, the solution obtained may or may not be feasible by overlapping considerations, hence it will be amended by the correction step, here the size of the circular container is updated (if it is needed) by means of a greedy approach. After the updates of the correction step a new iteration starts by switching the centres expressed in Cartesian coordinates to polar coordinates and vice-versa, the coordinates from this switching are considered as the initial solution

for current iteration, the process continue until the stopping criteria is met (number of iterations).

A.2 Correction step

Correction step procedures aim to detect possible overlaps among the circles and resolve them thereby obtaining at the end of each iteration of the heuristic algorithm a feasible solution. The correction step consist of two procedures: in the first procedure it resolves the overlaps found whilst in the second procedure the size of the container is updated. For the first procedure we use a greedy approach that we explain below.

A.2.1 Resolving overlaps through a greedy approach

As greedy approaches find local optima at every stage of the process, here we adapt that philosophy to resolve any overlap found. As we know every centre of a circle is defined by Cartesian coordinates (x_i, y_i) and at the same time by polar coordinates (r_i, θ_i) with the associated radius R_i for all $i = 1, ..., n$. To implement the greedy approach we consider all circle centres in their polar equivalent and consider them in ascending order, that is, the first circle to be investigated is the one whose polar coordinates (r_i, θ_i) with component r_i is the smallest, hence the closest one to the origin of the Cartesian plane. The investigation consists of detecting and resolving the overlaps that other circles may be creating with circle i. The position of circle i is fixed and if there are other circles overlapping with it, they are moved outward until circle i and the ones originally overlapping it are merely touching. Once we have resolved all the overlaps found with circle i we update the coordinates of all circles (except circle i because is the closest one) and continue the procedure with the next closest circle to the origin, we terminate this procedure when all circles have been investigated.

Taking Figure A.1 to explain and illustrate how we resolved the overlaps encountered, let us consider $r_i = \min_{1 \leq k \leq n}\{r_k\}$ the minimum component from the polar coordinates

and the correspondent Cartesian coordinates (x_i, y_i). To proceed to detect the overlaps we take the Euclidean distance between point (x_i, y_i) and all (x_j, y_j) $j \neq i, j = 1, ..., n$. If there is an overlap let us say, with circle j with coordinates (x_j, y_j) then we draw a line that passes through the origin with slope y_j/x_j, (in Figure A.1 this is the grey dotted line). To find the new coordinates for circle j we draw a circle with centre (x_i, y_i) with radius $R_i + R_j$, depicted in Figure A.1 as the blue dotted circle, the new coordinates$(\bar{x_j}, \bar{y_j})$ for circle j (the red one) will be defined by the intersection between the blue circle and the grey line. In other words, we move circle j outward along the ray from the origin through its centre until circle i and circle j are just touching.

Given this graphical description in technical terms, we just need to find the intersection between a circle and a line, in general the equation of a circle with centre (a, b) and radius r is given by $(x - a)^2 + (y - b)^2 = r^2$, for this particular case the centre of the blue circle is $(a = x_i, b = y_i)$, and radius $r = R_i + R_j$. Regarding for the equation of a line that passes through the origin is defined by $y = mx$, here the slope is given by $m = y_j/x_j$.

For ease of notation let us find the new coordinates of circle j of the centre (a, b) and radius r of blue circle and the slope of the grey line as m. Expanding the quadratic expression we have

$$x^2 - 2ax + a^2 + y^2 - 2by + b^2 = r^2 \tag{A.12}$$

by substituting $y = mx$ in equation A.12 we obtain

$$x^2 - 2ax + a^2 + (mx)^2 - 2b(mx) + b^2 = r^2 \tag{A.13}$$

which is reduced to

$$(1 + m^2)x^2 - 2(a + bm)x + (a^2 + b^2 - r^2) = 0 \tag{A.14}$$

The general solution of the quadratic equation A.14 is

$$x_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

where: $A = (1 + m^2)$, $B = -2(a + bm)$, $C = (a^2 + b^2 - r^2)$ then the two solutions are

$$(x_1, y_1) = (\tfrac{a+bm-t_1}{t_2}, \tfrac{m(a+bm-t_1)}{t_2})$$
$$(x_2, y_2) = (\tfrac{a+bm+t_1}{t_2}, \tfrac{m(a+bm+t_1)}{t_2})$$

where: $t_1 = \sqrt{-(b - am)^2 + (1 + m^2)r^2}$ and $t_2 = 1 + m^2$

From these two solutions [(x_1, y_1) and (x_2, y_2)]we keep the one whose polar component satisfy $\bar{r_j} > r_j$, in other words, given the new point $(\bar{x_j}, \bar{y_j})$ has polar coordinates $(\bar{r_j}, \theta_j)$. In this way circle j will be the red one and the distance between centre of circle i and circle j will be $R_i + R_j$.

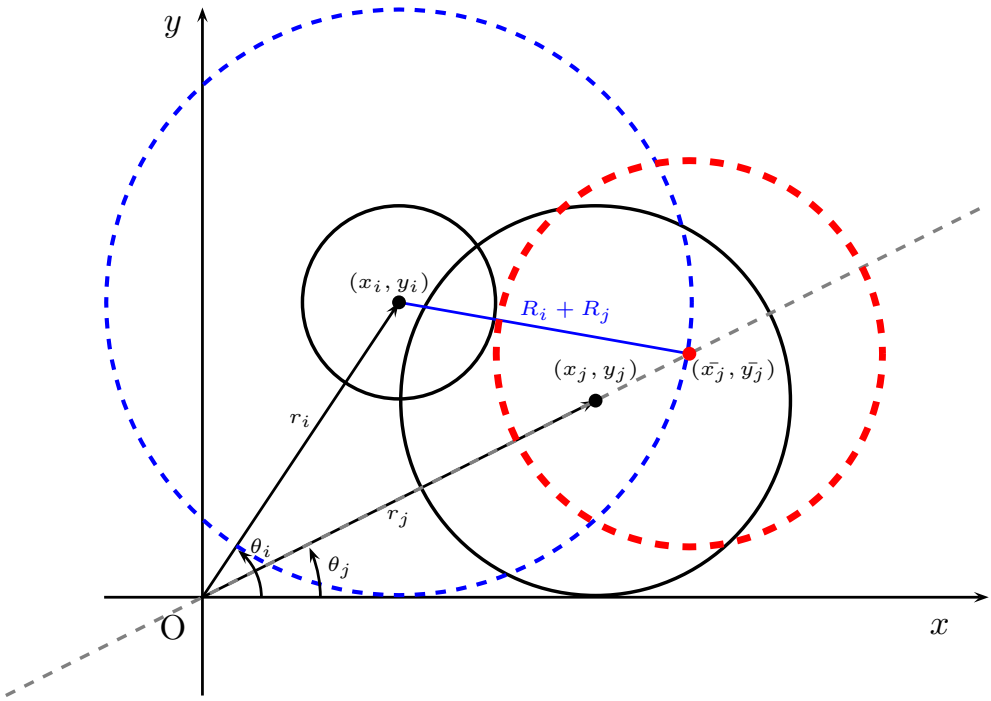


Figure A.1: Example of how the overlaps are resolved using the greedy approach

For the second part of the correction step, in order to determine the size (if it needs to be updated) of the container we consider the polar coordinates of all circles (r_i, θ_i) for all $i = 1, ..., n$. We take $R^* = \max_{1 \leq k \leq n}\{r_k + R_k\}$ as the current size of the circular

container corresponding to a feasible solution, where r_k is the polar coordinate associated with circle k and R_k the radius of circle k.

A.3 Computational results

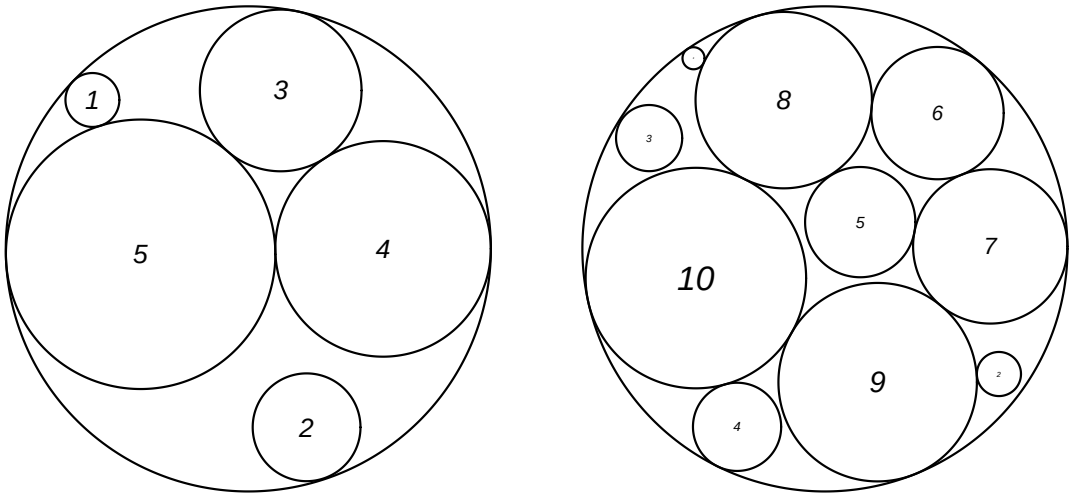
Regarding computational results we considered different values for the Δ, we first defined $\Delta_i = \frac{2R_i}{3R^*}$ as having a different value depending on the size of the circle. The results from that test were not as successful as we expected, in the light of that we decided to drop Δ. Essentially it means that all circles may overlap and also they are free to move, hence with n circles to be packed the elements of the Q set are all possible $(n(n-1)/2)$ pairs of circles (i, j) with $i \neq j$ and $i, j = 1, ..., n$. The best computational results are present in Table A.1, they were obtained considering 5 replications, each consisted of 50 iterations. For the test we considered instances with $n = 5, 10, 15, 20$ non-identical circles from the large variation instances ($R_i = i$). In Table A.1 we present the best-known results, our best results, the % deviation and the total time that our heuristic took to obtain the results.

Table A.1: Comparison with Packomania web site [72] for the circular container

n	Best-known solution	OUR Best solution	% deviation	Total time (min)
5	9.001397746	9.001392658	-0.000056522	1226.57
10	22.000193013	22.000193013	0.000000000	2564.63
15	38.837995508	39.151355478	0.806838678	6731.92
20	58.400567479	59.901665481	2.570348315	11282.67
Average			0.844282618	5451.45

From the bottom row of Table A.1 we can see that the average % deviation for the four instances considered is 0.844282618 with an average total time of 5451.45 minutes which is around 90.85 hours on average to obtain a solution. Although it seems that we

have good solutions for n=5, 10, for 15 and 20 non-identical circles the quality of the solutions are not as close to the best-known solution as we have expected. However, for $n = 5$ circles we improved over the best previously known and for $n = 10$ circles our results present same accuracy as the best-known. In Figure A.2 we show two pictures that represent the best solutions obtained using the greedy approach.



(a) 5 non-identical circles inside a circular container with radius 9.001392658

(b) 10 non-identical circles inside a circular container with radius 22.000193013

Figure A.2: Packing n non-identical circles with $R_i = i$ inside a circular container

A.4 Conclusions

In the light of the results presented here we decided not to pursue the approach outlined above. Instead we developed the scaled approach which was presented in Chapter 5.

References

[1] B. Addis, M. Locatelli and F. Schoen. *Efficiently packing unequal disks in a circle: a computational approach which exploits the continuous and combinatorial structure of the problem.* Operations Research Letters 2008;36:37–42.

[2] H. Akeb, M. Hifi and R. M'Hallah. *A beam search algorithm for circular packing problem.* Computers & Operations Research 2009;36:1513–1528.

[3] H. Akeb, M. Hifi and S. Negre. *An augmented beam search-based algorithm for the circular open dimension problem.* Computers & Industrial Engineering 2011; 61:373–381.

[4] H. Akeb and Y. Li. *A hybrid heuristic for packing unequal circles into a circular container.* International Conference on Service Systems and Service Management 2006;2:922–927.

[5] I. Al-Mudahka, M. Hifi and R. M'Hallah. *Packing circles in the smallest circle: an adaptive hybrid algorithm.* Journal of the Operational Research Society 2011; 62:1917–1930.

[6] A. Albano and G. Sapippo. *Optimal allocation of two-dimensional irregular shapes using heuristic search methods.* IEEE Transactions on Systems, Management and Cybernetics 1980;10(5):242–248.

[7] T. Aste and D. Weaire. *The pursuit of perfect packing.* Institute of Physics Publishing, 2000.

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

[9] E. Birgin and J. Gentil. *New and improved results for packing identical unitary radius circles within triangles, rectangles and strips*. Computers & Operations Research 2010;37(7):1318–1327.

[10] E. Birgin and F. Sobral. *Minimizing the object dimensions in circle and sphere packing problems*. Computers & Operations Research 2008;35(7):2357–2375.

[11] P. Bonami, L. Biegler, A. Conn, G.Cornuéjols, I. Grossmannn, C. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya and A. Wachter. *An algorithm framework for convex mixed integer nonlinear programs*. Discrete Optimization 2008;5(2):186–204.

[12] M. Bussieck, A. Drud and A. Meeraus. *Minlplib - a collection of test models for mixed-integer nonlinear programming*. Inform Journal on Computing 2003; 15(1):114–119.

[13] M. Bussieck and S. Vigerske. *MINLP solver software*. Wiley Encyclopaedia of Operations Research and Management Science, Wiley, New York, 2011 2011;URL http://www.math.hu-berlin.de/~stefan/minlpsoft.pdf.

[14] I. Castillo, F. Kampas and J. Pinter. *Solving circle packing problems by global optimization: Numerical results and industrial applications*. European Journal of Operational Research 2008;191(3):786–802.

[15] C. D'Ambrosio. *Application-oriented mixed integer non-linear programming*. Ph.D. thesis, University of Bologna, Italy, 2009. URL http://www.lix.polytechnique.fr/~dambrosio/DAmbrosio_Claudia_tesi.pdf.

[16] GAMS. *MINLP World*. URL www.gamsworld.org/minlp/minlplib.htm.

[17] J. George, J. George and B. Lamar. *Packing different-sized circles into a rectangular container*. European Journal of Operational Research 1995;84:693–712.

[18] P. Gill, W. Murray and M. Saunders. *User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming*, 2008. URL http://www.ccom.ucsd.edu/~peg/papers/sndoc7.pdf.

[19] R. Graham and B. Lubachevsky. *Dense packings of equal disks in an equilateral triangle: From 22 to 34 and beyond.* The Electronic Journal of Combinatorics 2 1995;URL http://www/emis.de/journals/EJC/Volume_2/PDFFiles/.

[20] R. Graham and B. Lubachevsky. *Repeated patterns of dense packings of equal disks in a square.* The Electronic Journal of Combinatorics 1996;3. URL www1.combinatorics.org/Volume_3/volume3.html#R16.

[21] P. Grassberger. *Prune-enriched rusenbluth method: Simulations of theta polymers of chain length up to 1000000.* Physical review 1997;56(3):3682–3693.

[22] I. Grossmannn. *Review of non-linear mixed integer and disjunctive programming techniques.* Optimization and Engineering 2002;3(3):227–252.

[23] A. Grosso, A. Jamali, M. Locatelli and F. Schoen. *Solving the problem of packing equal and unequal circles in a circular container.* Journal of Global Optimization 2010;47(1):63–81.

[24] A. Grosso, M. Locatelli and F. Schoen. *A population-based approach for hard global optimization problems based on dissimilarity measures.* Mathematical Programming Series A 2007;110(2):373–404.

[25] M. J. Haims and H. Freeman. *A multistage solution to the template-layout problem.* IEEE Transactions on Systems, Management and Cybernetics 1970;6:145–151.

[26] P. Hansen, N. Mladenović, J. Brimberg and J. Perez. *Handbook of Metaheuristics (Variable neighborhood search)*, volume 146 of *International Series in Operations Research & Management Science*. Springer, 2010.

[27] U. Hansmann and L. Wille. *Global optimization by energy landscape paving.* Physical Review Letters 2002;88(6)(Article 068105).

[28] A. Hertz, M. Plumettaz and N. Zufferey. *Variable space search for graph coloring.* Discrete Applied Mathematics 2008;156(13):2551–2560.

[29] A. Hertz, M. Plumettaz and N. Zufferey. *Corrigendum to "variable space search for graph coloring' [discrete appl. math. 156, 2551-2560 (2008)].* Discrete Applied Mathematics 2009;157(7):1335–1336.

[30] M. Hifi and R. M'Hallah. *Approximate algorithms for constrained circular cutting problems.* Computers & Operations Research 2004;31:675–694.

[31] M. Hifi and R. M'Hallah. *Adaptive and restarting techniques-based algorithms for circular packing problem.* Computational Optimization and Applications 2008; 39:17–35.

[32] M. Hifi and R. M'Hallah. *A dynamic adaptive local search algorithm for the circular packing problem.* European Journal of Operational Research 2008;183:1280–1294.

[33] M. Hifi and R. M'Hallah. *A literature review on circle and sphere packing problems: Models and methodologies.* Advances in Operations Research 2009;ID 150624. URL http://downloads.hindawi.com/journals/aor/2009/150624.pdf.

[34] M. Hifi, V. Paschos and V. Zissimopoulos. *A simulated annealing approach for the circular cutting problem.* European Journal of Operational Research 2004;159:430–448.

[35] K. Holmström, A. Göran and M. Edvall. *User's Guide for TOMLAB/SNOPT*, 2008. URL http://tomopt.com/docs/TOMLAB_SNOPT.pdf.

[36] W. Huang and M. Chen. *Note on: An improved algorithm for the packing of unequal circles within a large containing circles.* Computers & Industrial Engineering 2006; 50:338–344.

[37] W. Huang, Y. Li, C. Li and R. Xu. *New heuristics for packing unequal circles into a circular container.* Computers & Operations Research 2006;33:2125–2142.

[38] W. Huang, Y. Li and R. Xu. *A quasi-physical method of solving packing problem.* In *In proceedings of the 4th Metaheuristics International Conference (MIC'2001).* 2001; .

[39] W. Huang and T. Ye. *Greedy vacancy search algorithm for packing equal circles in a square.* Operations Research Letters 2010;38(5):378–382.

[40] W. Huang and T. Ye. *Quasi-physical global optimization method for solving the equal circle packing problem.* Science China Information Sciences 2011;54(7):1333–1339.

[41] W. Huang and S. Zhan. *A quasi-physical method for solving packing problems.* Acta Mathematicae Applicatae Sinica 1979;2(2):176–180.

[42] W. Huang and S. Zhan. *A quasi-physical method of solving packing problems.* Mathematical Reviews, American Mathematical Society 1982;.

[43] Y. Kochetov, P. Kononova and M. Paschenko. *Formulation space search approach for the teacher/class timetabling problem.* Yugoslav Journal of Operations Research 2008;18(1):1–11.

[44] G. N. L. Liberti and N. Mladenović. *Matheuristics: A good recipe for solving MINLPs*, volume 10 of *Annals of Information Systems*. Springer, 2010.

[45] R. Leary. *Global optimization on funneling landscapes.* Journal of Global Optimization 2000;18(4):367–383.

[46] S. Leyffer, J. Linderoth, J. Luedtke, A. Mahajan and T. Munson. *Minotaur solver*, 2012. URL http://wiki.mcs.anl.gov/minotaur/index.php/MINOTAUR.

[47] L. Liberti, G. Nannicini and N. Mladenović. *A recipe for finding good solutions to minlps.* Mathematical Programming Computation 2011;3(4):349–390.

[48] J. Liu, G. Li, D. Chen, W. Liu and Y. Wang. *Two-dimensional equilibrium constraint layout using simulated annealing.* Computers & Industrial Engineering 2010;59:530–536.

[49] J. Liu, G. Li and H. Geng. *A new heuristic algorithm for the circular packing problem with equilibrium constraints.* Science China Information Sciences 2011; 54(8):1572–1584.

[50] J. Liu, Y. Wang and J. Pan. *Efficiently packing circles into a larger containing circle.* In *Proceedings of the Second International Conference on High Performance Computing and Applications*, volume 5938 of *Lecture Notes in Computer Science*. Springer-Verlag, 2010; 250–256.

[51] J. Liu, S. Xue, Z. Liu and D. Xu. *An improved energy landscape paving algorithm for the problem of packing circles into a larger containing circle.* Computers & Industrial Engineering 2009;57(3):1144–1149.

[52] J. Liu, Y. Yao, Y. Zheng, H. Geng and G. Zhou. *An effective hybrid algorithm for the circles and spheres packing problems.* In *Combinatorial optimization and Applications, Third International Conference, COCOA 2009, Huangshan, China*, volume 5573 of *Lectures Notes in Computer Science*. Springer, 2009; 135–144.

[53] C. López and J. Beasley. *A note on solving minlp's using formulation space search.* Optimization Letters 2013;To appear.

[54] C. López and J. Beasley. *A heuristic for the circle packing problem with a variety of containers.* European Journal of Operational Research 2011;214(3):512–525.

[55] C. López and J. Beasley. *Packing unequal circles using formulation space search.* Computers & Operations Research 2013;40:1276–1288.

[56] Z. Lu and W. Huang. *Perm for solving circle packing problem.* Computers & Operations Research 2008;35:1742–1755.

[57] P. Machado and A. Leitão. *Evolving Fitness Functions for Mating Selection*, volume 6621 of *Lecture Notes in Computer Science*. Springer, 2011.

[58] H. Melissen. *Densest packings of congruent circles in a equilateral triangle.* The American Mathematical Monthly 1993;100(10):916–925.

[59] H. Melissen. *Acta mathematica hungarica.* Optimal packings of eleven equal circles in an equilateral triangle 1994;65(4):389–393.

[60] H. Melissen and P. Schuur. *Packing 16, 17 or 18 circles in an equilateral triangle.* Discrete Mathematics 1995;145:333–342.

[61] H. Mittelmann. *Performance of commercial and noncommercial optimization software.* In *INFORMS 2012*. Phoenix, Arizona, USA, 2012; URL http://plato.asu.edu/talks/phoenix.pdf.

[62] N. Mladenović, F. Plastria and D. Urošević. *Reformulation descent applied to circle packing problems.* Computers & Operations Research 2005;32(9):2419–2434.

[63] N. Mladenović, F. Plastria and D. Urošević. *Formulation space search for circle packing problems. in "Engineering Stochastic Local Search Algorithms. Designing,*

Implementing and Analyzing Effective Heuristics", Proceedings of the International Workshop, SLS 2007, Brussels, Belgium, September 6-8, 2007. In *Lecture Notes in Computer Science*, volume 4638. 2007; 212–216.

[64] A. Müller, J. Scheider and E. Schömer. *Packing a multidisperse system of hard disks in a circular environment.* Physical Review 2009;79:021102.

[65] M.R.Bussieck and A. Pruessner. *Mixed-integer nonlinear programming.* SIAG/OPT Viwes-and-News 2003;14(1):19–22.

[66] E. Pardo, N. Mladenović, J. Pantrigo and A. Duarte. *Variable formulation search for the cutwidth minimization problem.* Applied Soft Computing 2013;To appear.

[67] P.G.Szabo, M. Markot and T. Csendes. *Global optimization in geometry - circle packing into the square.* Essays and Survays in Global Optimization 2005;.

[68] J. Pintér. *Nonlinear optimization with GAMS/LGO.* Journal of Global Optimization 2007;38(1):79–101.

[69] B. K. R. Fourer, D.M. Gay. *AMPL A modeling language for mathematical programming.* Duxbury, 2003. URL http://www.ampl.com/.

[70] R. S. Rajab. *Some applications of continuous variable neighbourhood search meta-heuristic (mathematical modelling).* Ph.D. thesis, Brunel University, 2011.

[71] C. Sammut. *Encyclopedia of machine learning.* URL http://www.springerreference.com/docs/html/chapterdbid/178780.html.

[72] E. Specht. *Packomania*, 1999. URL http://www.packomania.com.

[73] Y. Stoyan and G. Yaskov. *Mathematical model and solution method of optimization problem of placement of rectangles and circles taking into account special constraints.* International Transactions in Operational Research 1998;5(1):45–57.

[74] Y. Stoyan and G. Yaskov. *A mathematical model and a solution method for the problem of placing various-sized circles into a strip.* European Journal of Operational Research 2004;156:590–600.

[75] Y. Stoyan and G. Yaskov. *Packing equal circles into a circle with circular prohibited areas*. International Journal of Computer Mathematics 2012;89(10):1355–1369.

[76] P. Szabó, M. Markót, T. Csendes, E. Specht, L. Casado and I. Garcia. *New approaches to circle packing in a square: with program codes*, volume 6. Springer, 2007.

[77] S. Vigerske, 2013. Private communication.

[78] D. Wales and J. Doye. *Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms*. The Journal of Physical Chemistry A 1997;101(28):5111–5116.

[79] H. Wang, W. Huang, Q. Zhang and D. Xu. *An improved algorithm for the packing of unequal circles within a large containing circles*. European Journal of Operational Research 2002;141:440–453.

[80] D. Zhang and A. Deng. *An effective hybrid algorithm for the problem of packing circles into a large containing circle*. Computers & Operations Research 2005; 32:1941–1951.

[81] D. Zhang and W. Huang. *A simulated annealing algorithm for the circles packing problem*. Lecture Notes in Computer Science 2004;3036:206–214.