UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica

EVOLUTIONARY COMPUTATION SOLUTIONS TO THE CIRCLE PACKING PROBLEM

TESIS

Que para obtener el grado de:

DOCTOR EN CIENCIAS EN INGENIERÍA ELÉCTRICA

Presenta:

JOSÉ MARTÍNEZ PEÑA

Director de Tesis

Dr. Juan José Flores Romero

Doctor en Ciencias

Co-Director de Tesis

Dr. Felix Calderón Solorio

Agosto 2015.

Morelia, Michoacán

EVOLUTIONARY COMPUTATION SOLUTIONS TO THE CIRCLE PACKING PROBLEM

Los Miembros del Jurado de Examen de Grado aprueban
la **Tesis de Doctorado en Ciencias en Ingeniería Eléctrica Opción en Sistemas Computacionales** de ***José Martínez Peña***

Dr. Jaime Cerda Jacobo
Presidente del Jurado

Dr. Juan José Flores Romero
Director de Tesis

Dr. Félix Calderón Solorio
Co-director

Dr. José Antonio Camarena Ibarrola
Vocal

Dr. Hugo Jair Escalante
Revisor Externo (INAOE)

Dr. Félix Calderón Solorio
Jefe de la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica. UMSNH
(Por reconocimiento de firmas)

UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO
Agosto 2015

Acknowledgements

Dr. Juan José Flores Romero
Por ser mi guía y todo su apoyo a los largo de este camino

Dr. Félix Calderón Solorio
Por su valiosa aportación y apoyo para llegar a culminar este trabajo

List of Publications

JCR Journal Papers

"Evolutionary Computation Solutions to the Circle Packing Problem"
Juan J. Flores, José Martínez, and Félix Calderón.
Soft Computing. A Fusion of Fundations, Methodologies and Applications. ISSN 1432-7643. February, 2015. DOI 10.1007/s00500-015-1603-y. Springer-Verlag Berlin Heidelberg 2015.

ISI-Thomson Conference Papers

"Performance Comparison of Repair Heuristics for the Circle Packing Problem"
Juan J. Flores, José Martínez, and Félix Calderón.
In Book: MICAI 2014: Artificial Intelligence. Editors: Alexander Gelbukh, Félix Castro Espinoza, Sofía N. Galicia-Haro. IEEE Computer Society Order Number P5521. ISBN 978-1-4799-9900-2. Pages: 183−189. Library of Congress Number 2015935989. BMS Part Number CFP1434B-PRT. Los Alamitos, CA 90720

"Solving a Scholar Timetabling Problem Using a Genetic Algorithm-Study Case: Instituto Tecnológico de Zitacuaro"
Noel Rodriguez, José Martínez, Juan J. Flores, and Mario Graff.
In Book: MICAI 2014: Artificial Intelligence. Editors: Alexander Gelbukh, Félix Castro Espinoza, Sofía N. Galicia-Haro. IEEE Computer Society Order Number P5521. ISBN 978-1-4799-9900-2. Pages: 197−202. Library of Congress Number 2015935989. BMS Part Number CFP1434B-PRT. Los Alamitos, CA 90720

Resumen

En esta tesis se presenta una solución basada en Computación Evolutiva al Problema del Empaquetamiento de Círculos (ECPP). El Problema del Empaquetamiento de Círculos consiste en colocar un conjunto de círculos en un contenedor sin que existan traslapes entre los círculos; un conocido problema que es NP-duro. Dada la imposibilidad de resolver este problema de manera eficiente, se han propuesto métodos tradicionales y heurísticos para resolverlo. Una representación nativa de cromosomas en una búsqueda heurística basada en la población conduce a altas probabilidades de violación de las restricciones del problema, es decir, provoca traslapes. Para convertir las soluciones que violan las restricciones en las que no (es decir, soluciones factibles), en esta tesis se proponen dos mecanismos de reparación. El primero considera cada círculo como un anillo elástico; los traslapes existentes crean fuerzas de repulsión que llevan los círculos a los espacios donde se resuelven estos traslapes. La segunda forma esta basada en la Triangulación de Delaunay con los centros de los círculos y las reparaciones de los círculos en cada triángulo a la vez, asegurando triángulos reparados que no serán modificados en el futuro. Basados en las heurísticas de reparación propuestas, se presentan los resultados de la solución al Problema del Empaquetamiento de Círculos para un conjunto de problemas de círculos unitarios (cuyas soluciones óptimas son conocidas). Estos problemas con óptimas soluciones conocidas se resuelven usando Algoritmos Genéticos, Estrategias Evolutivas, Optimización por Enjambre de Partículas y Evolución Diferencial. El rendimiento de las soluciones se comparan con las soluciones conocidas basadas en la densidad del empaquetamiento. Luego se realizo una serie de experimentos para determinar el rendimiento de ECPP con círculos no unitarios. En primer lugar, se comparan los resultados del ECPP con las de un concurso público, que se destacan como el récord mundial para ese caso particular del CPP para círculos no unitarios. En un segundo conjunto de experimentos, se controla la varianza del tamaño de los círculos. En todos los experimentos ECPP produce soluciones casi óptimas satisfactoriamente.

Palabras clave: Optimización, Computación Evolutiva, Triangulación de Delaunay.

Abstract

In this thesis I present an evolutionary computation based solution to the Circle Packing Problem (ECPP). The Circle Packing Problem consists of placing a set of circles into a container without overlaps; a problem known to be NP-hard. Given the impossibility to solve this problem efficiently, traditional and heuristic methods have been proposed to solve it. A naïve representation for chromosomes in a population-based heuristic search leads to high probabilities of violation of the problem constraints, i.e. overlapping. To convert solutions that violate constraints into ones that do not (i.e. feasible solutions), in this thesis I propose two repair mechanisms. The first one considers every circle as an elastic ring; overlaps create repulsion forces that lead the circles to positions where the overlaps are resolved. The second one forms a Delaunay Triangulation with the circle centers and repairs the circles in each triangle at a time, making sure repaired triangles are not modified later on. Based on the proposed repair heuristics, I present the results of the solution to the CPP problem to a set of unit circle problems (whose exact optimal solutions are known). These benchmark problems are solved using Genetic Algorithms, Evolutionary Strategies, Particle Swarm Optimization, and Differential Evolution. The performance of the solutions are compared to those known solutions based on the packing density. I then perform a series of experiments to determine the performance of ECPP with non-unitary circles. First, I compare ECPP's results to those of a public competition, which stand as the world record for that particular instance of the non-unitary CPP. On a second set of experiments, I control the variance of the size of the circles. In all experiments ECPP yields satisfactory near-optimal solutions.

Keywords: Optimization, Evolutionary Computation, Delaunay Triangulation.

Contents

List of Figures

List of Tables

Acronymous

AI	**A**rtificial **I**ndividuals
BS1	**B**eam **S**earch **1**
BS2	**B**eam **S**earch **2**
CH	**C**onvex **H**ull
CODP	**C**ircular **O**pen **D**imension **P**roblem
CPP	**C**ircle **P**acking **P**roblem
DT	**D**elaunay **T**riangulation
DE	**D**ifferential **E**volution
EAs	**E**volutionary **A**lgorithms
EC	**E**volutionary **C**omputation
ECPP	**E**volutionary **C**ircle **P**acking **P**roblem
ES	**E**volutionary **S**trategies
ESCO	**E**volution **S**trategy **C**rossover **O**perator
GA	**G**enetic **A**lgorithms
ILP	**I**somorphic **L**ayout **P**attern
LGO	**L**ipschitz **G**lobal **O**ptimizer
LP	**L**inear **P**rograming
LPPSO	**L**ayout **P**attern-Based **P**articular **S**warm **O**ptimization
MBS	**M**odified **B**illiard **S**imulation
NLP	**N**on-isomorphic **L**ayout **P**attern
NP	**N**ested **P**artitioning
NP-hard	**N**o **P**olynomial hard
NP-complete	**N**o **P**olynomial complete
PECS	**P**acking **E**qual **C**ircles in a **S**quare
PERM	**P**runed **E**nriched **R**osenblueth **M**ethod

PSO	**P**article **S**warm **O**ptimization
RD	**R**eformulation **D**escent
SASS	**S**ingle **A**gent **S**tochastic **S**earch
TAMSASS	**T**hreshold **A**ccepting **M**odified **S**ingle **A**gent **S**tochastic **S**earch
TS	**T**abu **S**earch
UCPP	**U**nit **C**ircle **P**acking **P**roblem

List of Symbols

a_0	area of rectangular container
C	set of circles
c_0	enclosing circle
c_0^*	enclosing circle of solution
c_i	i-th circle
H_0	null hypothesis
H_1	alternative hypothesis
L	length of rectangular container
N	number of circles
P	central points of circles
d_P	fraction of the container area
r_0	radius of enclosing circle
r_0^*	$radius^*$ of enclosing circle
r_i	radius of circle i
S_1	side 1 of rectangular container
S_2	side 2 of rectangular container
S_3	side 3 of rectangular container
S_4	side 4 of rectangular container
W	with of rectangular container
(x, y)	coordinates
(x_0, y_0)	coordinates of enclosing circle
(x_i, y_i)	coordinates of circle i
(x_i^*, y_i^*)	coordinates of circle i of optimal solution

$\mathbb{R}$	real numbers
ρ	density of solution
ρ^*	optimal density of solution

Chapter 1

Introduction

Given N circles, of given radii, the Circle Packing Problem (CPP) is concerned with how to pack those circles into a container, without overlapping. CPP has a wide spectrum of applications; it is encountered in a variety of real world applications, including production and packing for the textile, apparel, naval, automobile, aerospace, and food industries Castillo et al. [2008].

Many variations features exist on this problem; e.g., the container can be a circle, rectangle, or polygon, and the objects can be a circular, rectangular, or irregular. This thesis addresses CPP, where the objects are circles and container are circles, squares or rectangulars. This problem has been proven to be NP-hard Demaine et al. [2010]. So, heuristic search methods are generally proposed to solve this problem.

Packing circular objects is a challenge in discrete and computational geometry Szabó et al. [2007], with a large number of circular objects to pack, the optimal solution is very difficult to find. An optimal solution may be rotated, reflected, or the circular objects reordered; hence, the number of equivalent optimal solutions blows up as the number of circular objects increases Hifi and M'Hallah [2009a]. In addition, one or more of the circular objects may be moved slightly without affecting the optimal solution. In fact, there exists a continuum of optimal solutions Hifi and M'Hallah [2009a].

CPP has many important applications in manufacturing, logistics, networks, facility layout, and materials science Castillo et al. [2008]. For example, in the automobile industry, design engineers have to estimate the size of the hole to be drilled on the body of a car and through which they plan to pass a bundle of wires that connect the car's

sensors to the dashboard Sugihara et al. [2004]. The hole has to be large enough to allow all wires to pass, but as small as possible to avoid unnecessarily weakening the body Sugihara et al. [2004]. CPP is also encountered in the manufacturing of sprockets for the motorcycle industry Dowsland et al. [2007]. Similarly, it is of interest to the telecommunication, electrical, oil companies, and refineries, which have to pass bundles of different types of cables, pipes, and insulated pipes through cylindrical shapes over very long distances. The smaller the diameters of the cylinders, the cheaper the cost is. Finally, CPP emerges in material science where it is used to interpret topological relationships encountered when analyzing the normal grain growth in two dimensions Nordbakke et al. [2004] and to model certain absorption patterns of molecules Harary et al. [1996].

Last, there is the issue of computational accuracy. The goal is to search for the best packing of the N circles inside the container c_0, where the best packing minimizes unused space.

This thesis presents an approach to get a feasible solution through evolutionary computation. This approach is called EC-CPP (for Evolutionary Computation CPP). Perhaps the first chromosome representation that comes to mind is a vector containing the coordinates of the center of each circle. Most Evolutionary Computation methods start with a random population, and then successively apply perturbations (i.e., genetic operators) to members of the population, until the best possible solution is reached. When generating new solutions, both randomly or by perturbations, the probability of generating overlaps is high.

Two methods have been extensively explored and used to enable metaheuristic optimization to produce solutions to constrained problems. One is to apply a penalty function to individuals that violate constraints. A Penalty function is normally expressed as a term or coefficient in the fitness function that makes violating individuals the least fit, so they are discarded in the evolutionary process. This approach has two main drawbacks. The first one is that the design of the penalty function is domain dependent and non-trivial, and the second one is the time spent in processing a very large number of violating individuals which end up being discarded. The second approach deals with constrained optimization, known as repairing, which uses a function that takes an individual that violates the constraints and returns a different individual that does not violate the constraints (the closer to the original, the better). This is the approach used in this thesis.

EC-CPP implements two repair heuristics and use them in conjunction with several metaheuristic search methods. I evaluate the results EC-CPP produces in terms of the obtained density; i.e., the ratio between the sum of the areas of the circles inside it and the area of the container, and compare them with the existing benchmarks Specht [1999]. Experimental results show that our approach has a good performance in terms of the solutions' densities. CPP can be divided in two variations of the same problem. The first one known as the unit circle packing problem (UCPP) considers circles of the same size. The second and more general one considers circles of arbitrary sizes. I have empirically tested EC-CPP with many instances of both variations of the CPP. Nevertheless, our evaluations focus on the UCPP, given that there are theoretical solutions for the problem up to 1104 circles.

1.1 Problem Definition

Consider a set C of N circles, where each circle $c_i \in C$, has a radius $r_i \in \mathbb{R}$.

1.1.1 The Circle Packing Problem in Circular Container

CPP consists of providing the locations of the circles' centers (x_1, y_1), (x_2, y_2), ..., (x_N, y_N), that minimize the radius r_0 and center (x_0, y_0) of the containing circle c_0, subject to:

$$(x_0 - x_i)^2 + (y_0 - y_i)^2 \leq (r_0 - r_i)^2, i \in \{1, 2, \ldots, N\} \tag{1.1}$$

$$(x_i - x_j)^2 + (y_i - y_j)^2 \geq (r_i + r_j)^2, i \neq j, \forall (i, j) \in \{1, 2, \ldots, N\}^2 \tag{1.2}$$

where constraint (1.1) stablishes that all circles must be inside the container circle c_0 and constraint (1.2) stablishes that circles must not overlap.

It is important to note that the smaller the radius of the container circle, the less unused space, therefore, the greater the packing density. Although the problem has been stated as a minimization of Eq. (1.1), where the objective function is directly the radius of the enclosing circle, it could be formulated as a maximization problem, where the objective function is the packing density. Both optimization problems are equivalent.

If (x_i^*, y_i^*) $i \in \{1, \ldots, N\}$ is an optimal solution to a CPP instance, and r_0^* is the radius of the enclosing circle c_0^*, the density of the solution is:

$$\rho^* = \frac{\sum_{i=1}^{N} {r_i}^2}{{r_0^*}^2}. \tag{1.3}$$

Any near-optimal solution (x_i, y_i) $i \in \{1, \ldots, N\}$, where $r_0 > r_0^*$, has a density of 1.3. Since $r_0 > {r^*}_0$, $\rho > \rho^*$. Thus, minimizing r_0 is equivalent to maximizing ρ.

1.1.2 The Circle Packing Problem in Rectangular Container

CPP in Rectangle consists of providing the locations P of the circles' centers (x_1, y_1), (x_2, y_2), ..., (x_N, y_N), that maximize the density d_P of the rectangular container. Suppose a rectangular container of given width W and length L, and a finite set C of N circles with not necessarily equal radii. The container is embedded in the Euclidian plane. The density d_P measures the fraction of the container area covered by circles and is defines as:

$$d_P = \frac{1}{LW} \sum_{i=1}^{N} \pi r_i^2 = \frac{\pi}{LW} \sum_{i=1}^{N} r_i^2 \tag{1.4}$$

The problem is to determine an optimal packing for the set C to maximize d_P subject to:

$$(x_i - x_j)^2 + (y_i - y_j)^2 \geq (r_i + r_j)^2, i \neq j, \forall (i, j) \in \{1, 2, \ldots, N\}^2 \tag{1.5}$$

$$d_{i,S1} = x_i - r_i \geq 0, \forall i \in \{1, 2, \ldots, N\} \tag{1.6}$$

$$d_{i,S2} = W - y_i - r_i \geq 0, \forall i \in \{1, 2, \ldots, N\} \tag{1.7}$$

$$d_{i,S3} = y_i - r_i \geq 0, \forall i \in \{1, 2, \ldots, N\} \tag{1.8}$$

$$d_{i,S4} = L - x_i - r_i \geq 0, \forall i \in \{1, 2, \ldots, N\} \tag{1.9}$$

where constraint (1.5) stablishes that circles must not overlap. Contraints (1.6) to (1.9) assure that all packed circles lie completely within the container.

1.2 Levels of Abstraction

When solving CPP using a population-based metaheuristic, individuals are produced either at random or by perturbations generated by (random) combination of information encoded in other individuals. Under those circumstances, the probability of generating individuals that do not comply with the constraints (i.e., at least a pair of circles overlap) is very high. I distinguish two types of individuals; those that do not violate the constraints form the feasible search space, or feasible space; and the set of all possible individuals, regardless of whether or not they violate the constraints, form the general search space, or search space.

To solve CPP using a population-based metaheuristic, I need to explore the search space and determine an individual that does not violate the constraints and produces the smallest possible radius of the enclosing container (the fitness or objective function).

There are two general approaches to deal with individuals that represent unfeasible solutions: penalty functions and repair functions.

A penalty method discourages unfeasible solutions by giving penalties so that feasible solutions are preferred to unfeasible solutions. Michalewicz Michalewicz [1996] has a good summary on constraint handling methods for evolutionary algorithms, and most of the existing methods are based on penalty functions. Each method is different in the amount of penalty assigned to unfeasible individuals. These variations become problem dependent for better performance.

Repair functions modify unfeasible solutions to produce feasible ones. This mapping preferably has to find the closest feasible solution to the individual to be repaired: a repaired solution substitutes the unfeasible solution and can be used for the search process. There are no general guidelines on how to repair unfeasible solutions. Most of the repair heuristics are problem dependent.

At the Feasible Space, metaheuristics from the field of Evolutionary Computation such as Genetic Algorithms (GA), Evolutionary Strategies (ES), Differential Evolution (DE), and Particle Swarm Optimization (PSO) can be applied, to optimize the objective function i.e., find the smallest possible container circle. These metaheuristics do not always guarantee an optimal solution. However, in most cases they give a near optimal solution

with less effort and time than the mathematical methods.

In this thesis we propose the use of two different repair functions. Repair functions allow us to view the search process at two levels of abstraction. At the lower level we have the search space; at the higher level we have the feasible space (see Fig.1.1). Some individuals in the search space can be mapped to the feasible space, but not all of them. The initial population will be composed, in its great majority, of individuals that do not comply to the non-overlapping constraints. Those individuals are repaired, so that the process starts with a population of individuals that can be mapped to the feasible space. The application of perturbations (a. k. a. genetic operators) to feasible individuals, will most likely produce unfeasible individuals. Offsprings are also repaired.

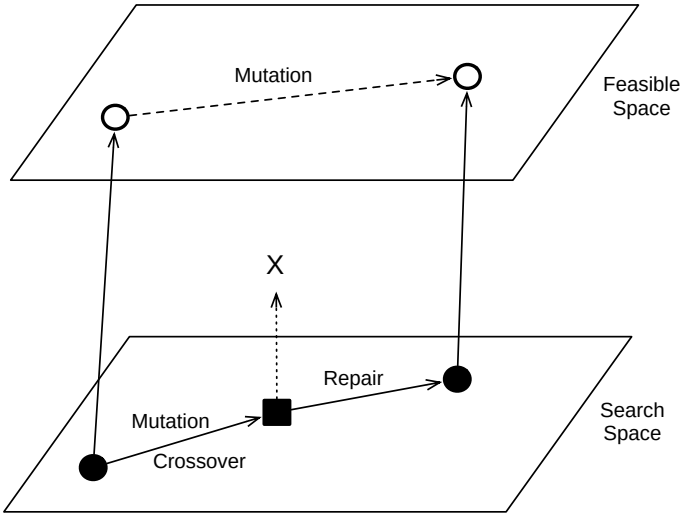


FIGURE 1.1: Feasible Space

1.3 CPP definition revisited

Given the above, we can view the evolutionary process as occurring at the feasible space as an unconstrained optimization problem. Now, from the feasible space level of abstraction, CPP can be formally stated as follows:

Given a set of circles C, determine these circles' positions (x, y), such that minimize the objective function

$$r_0^* = \min_{r_0} f(C) \quad for \;\; circular \;\; container \tag{1.10}$$

$$LW^* = \min_{L,W} f(C) \;\; for \;\; rectangular \;\; container \tag{1.11}$$

where r_0, the radius of the container circle c_0, and where LW, the area of the container rectangular c_0.

1.4 Justification

The mathematical model that represents CPP consists mainly of two types of constraints: the container boundary and the non-overlapping constraints; a problem known to be NP-hard, which justifies heuristic methods for its resolution. This problem has not been solved satisfactory using Evolutionary Computation.

The container boundary constraint ensures that all circles lie inside the container and the non-overlapping constraint ensures that for any two given circles, the distance between their centers are at least the sum of their radii. When both sets of constraints are satisfied we have a feasible configuration for the packing problem. Due to the complexity to obtain a feasible solution near to the optimal, we propose two repair mechanism to make feasible individuals.

1.5 About this Document

The remainder of this thesis is organized as follow: Chapter 2 gives a literature survey of work related to the circle packing problem. Chapter 3 describes the proposed population-based solution. Chapter 4 presents the computational results and some examples. The conclusions are presented in Chapter 5. Finally, the best obtained results on graphic form are presented in Appendix A.

Chapter 2

Related Work

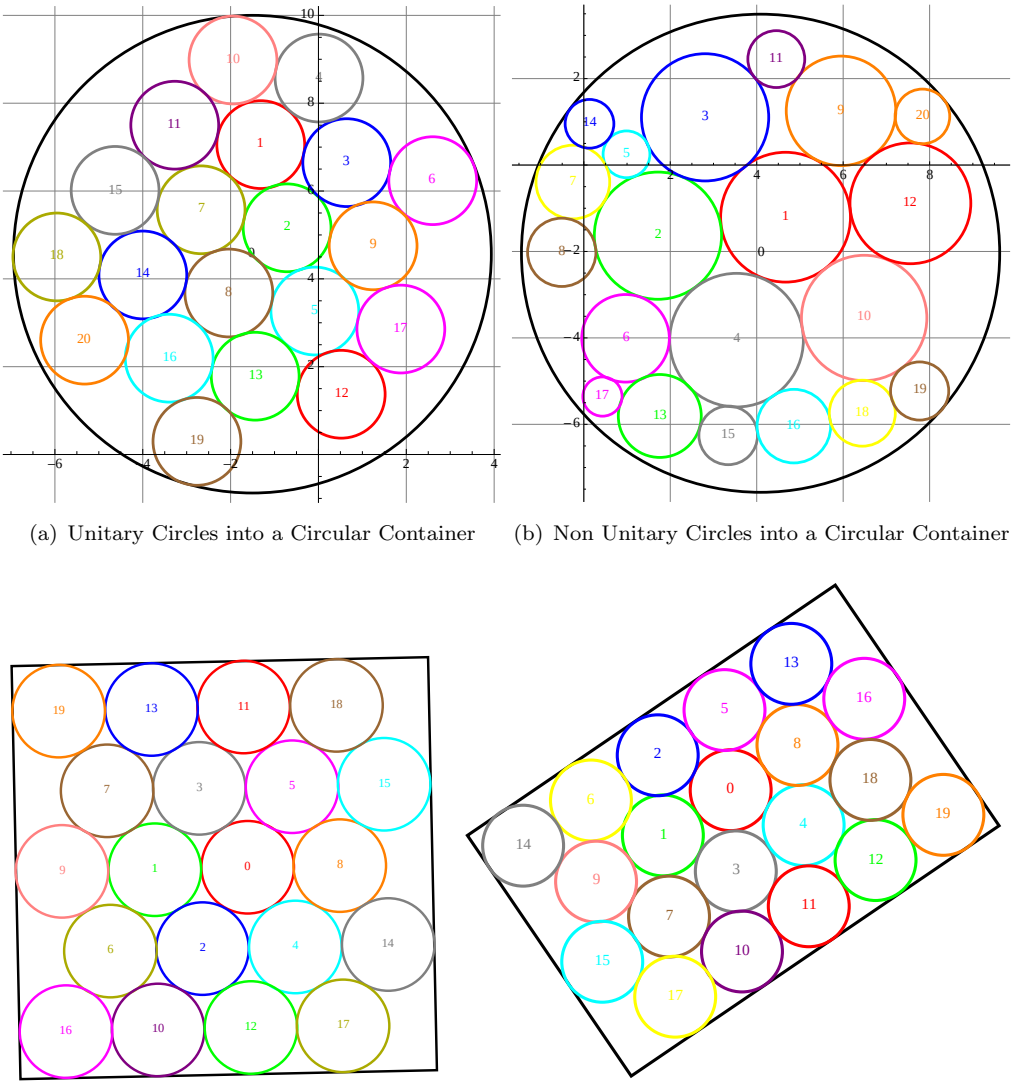
This literature review has the intention to give a general idea of the methods that have been used to tackle the circle packing problem in recent years. Castillo et al. [2008] and Hifi and M'Hallah [2009b] present a review of the packing problem considering circular and rectangular containers. In Castillo et al. [2008] authors present several circle packing problems, review their industrial applications, and some exact and heuristic strategies for their solution. They also present illustrative numerical results using generic global optimization software packages. In Hifi and M'Hallah [2009b] the authors present a detail review of what they called the most relevant literature on efficient models an methods for packing circular objects into Euclidean plane regions.

2.1 Circle Packing Problems

CPP is concerned with the arrangement of a finite number of circles inside of enclosing container without overlaps. In the following sections we will highlight the work that, to our consideration, is relevant to our research. There are many different ways to approach the packing problem: by method used, by the size of the circles being unit or not, by the shape of the container, etc. This section distinguishes between two main approaches to solve this problem: non-evolutionary solutions and evolutionary computation.

Regarding the size of the circles to be packed the problem is categorized as that of "packing identical circles" when all 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 presents four examples of the packing problem. Figure 2.1(a) shows 20 unit circles inside a circular container, Figure 2.1(b) presents 20 non-unit circles inside a circular

container. Figure 2.1(c) illustrates 20 unit circles inside a square container and Figure 2.1(d) presents 20 unit circles inside a rectangular container.



(a) Unitary Circles into a Circular Container (b) Non Unitary Circles into a Circular Container

(c) Unitary Circles into a Square Container (d) Unitary Circles into a Rectangular Container

FIGURE 2.1: Packing $n = 20$ circles into Circular, Square, and Rectangular Containers

2.2 Non-evolutionary Solutions

Wang et al. [2002] describe quasi-physical, quasi-human algorithm that mimics the human behavior to avoid being trapped into a local minimum. This method is an analogy to the physical model in which a number of smooth cylinders are packed inside a container. A strategy is proposed to trigger a jump for a stuck object in order to get out of local minima. The algorithm can be assimilated to an adaptive Tabu search. It randomly generates an initial pattern where every circle has its center inside the containing

circle. It measures the infeasibility of a solution, and translates a circle, whose position is infeasible, along both axes. The translation distance is a function of an adaptive step size and the gradient of the objective function of the current pattern. Since the iterative process converges quickly to feasible local optima, circles are allowed to jump in search for a new position within the containing circle.

Wenqi and Yan [2004] formulate CPP as a potential energy function by simulating a system of elastic solids. They position all circles randomly inside the containing circle. If this configuration has no overlapping circles, a feasible solution is at hand. Otherwise, the elastic repulsion forces generated by the overlaps drive the overlapping circles to restore their shape and size. The circles move along straight lines, colliding with each other and with the containing circle until the composition of elastic forces is decreased to zero. If the amount of overlap is also decreased to zero, then the process stops with a feasible solution. Otherwise, the process restarts.

Stoyan and Yas'kov [2004] present a mathematical model of the problem of placing different-radii circles into a strip considering several peculiarities. On the ground of these preculiarities, an original method of transition from one local minimum to another to provide a decrease of the objective function value is suggested. The method is based on the idea of increasing the problem dimension and a reduced gradient method, as well as on the concept of active inequalities and the Newton method. The authors consider a strip container of fixed width aiming to minimize 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.

Hifi et al. [2004] propose a heuristic for the constrained and the unconstrained circular cutting problem based upon simulated annealing. They define an energy function, the small values of which provide a good concentration of the circular pieces on the left bottom corner of the initial rectangle. Such values of energy correspond to configurations where pieces are placed in the rectangle without overlapping. The approach used is mainly based upon an energy function that, when it becomes minimum, it provides

solutions composed by a set of pieces concentrated at the bottom-left corner of the initial rectangle. They compare their work against a modified version of the original algorithms presented in Albano and Sapuppo [1980], Haims and Freeman [1970] 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 Stoyan and Yas'kov [2004] and to those extracted somehow in Graham and Lubachevsky [1996]. Their computational results are presented in terms of the percentage coverage of the area of the rectangular container. Computational results show that the approach presented is able to produce good solutions when compared with different approaches of the literature.

Szabó et al. [2005] present a review paper that summarizes the research work mostly done by the authors on packing equal circles in the unit square in the last years, 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, billiard simulation, perturbation method, TAMSASS-PECS (Threshold Accepting Modified Single Agent Stochastic Search for Packing Equal Circles in a Square) method, a deterministic approach based on LP-relaxation and MBS (Modified Billiard Simulation) algorithm. The energy function minimization: for this approach the objective function can be interpreted as a potential or energy function. A physical analogy of this approach is to regard the points as electrical charges (all positive or all negative) which are repulsing each other. If the minimal distance between the charged particles increases, the corresponding value of the potential function decreases. The billiard simulation method is physically motivated too. Let us consider a random arrangement of points. Draw equal circles around the points without overlapping. Each circle can be considered as a ball with an initial radius, moving direction and speed. Start the balls and increase slowly the common radius of them. The swing of each ball during the process will be less and less. The algorithm stops when the packing or a substructure of 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 center of the circle within a distance 0.25 and the location of the nearest neighbor is updated if the distance between them has increased, this process is repeated until there are no more circles to move. The TAMSASS-PECS method is based on the Threshold Accepting global optimization technique and a modified SASS local optimization algorithm. The algorithm starts with a pseudorandom initial packing, a standard deviation and with a threshold level. The algorithm improve the current solution by an iterative procedure. At every step it tries to find a better position of the actual point using a local search. The stopping criterion is based on the value of the standard deviation, which is decreased at every iteration. The framework of the method is the Threshold Accepting

approach. It is a close alternative of the Simulated Annealing algorithms. It accepts every move that leads to a new approximate solution not much worse than the current one and rejects other moves. A deterministic approach based on LP-relaxation: for this approach the circle packing problem can be regarded as an all-quadratic optimization problem, i.e. an optimization problem with not necessarily convex quadratic constraints. The hardness is due to the large number of constraints. This approach provides a rectangular subdivision branch-and-bound algorithm. To give an upper bound at each node of the branch-and-bound tree is used the special structure of the constraints and gave an LP-relaxation. Finally the MBS (Modified Billiard Simulation) algorithm distributes randomly n points inside the unit square and blow them up in a uniform manner. This can be done by incrementing the radii gradually from an initial value of $r_0 = \sqrt{\frac{10}{23n\pi}}$. In early stages of the process, when the distance between the small circles is much greater than their size and no collisions occur, there is no need to change their positions. As the circles grow, they have to deal with collisions. During the process when the decrease is too small or the number of iterations is larger than a given number, the calculation stops.

Mladenovic et al. [2005] apply a general Reformulation Descent heuristic (RD) to the problem of identifying the largest radius of identical circles that can be packed into a unit containing circle. RD iterates switching from solving CPP expressed in Cartesian coordinates to solving it expressed in polar coordinates and viceversa until no further improvement is obtained.

Zhang and Deng [2005] adopt the model of Wang et al. [2002], and use a hybrid approach consisting of simulated annealing to explore the neighborhood of the current solution, and tabu search to implement the jumps. When exploring the neighborhood of the current solution, one of the circles whose position is infeasible is translated and the degree of infeasibility of the neighbor is computed. A neighboring solution that reduces the degree of infeasibility becomes the incumbent solution whereas a nonimproving solution is accepted with a given probability, which decreases as the search becomes more selective.

Huang and Chen [2006] propose an improved version of the Wang's algorithm Wang et al. [2002] for solving CPP with equilibrium constraints. An efficient strategy of accelerating the search process is introduced in the gradient descent method.

Akeb et al. [2009] solve CPP using a binary search to determine r, and a beam search to check the feasibility of packing the n circles into C, when the radius is r. A node of level l, $l = 1, \ldots, N$, of the beam search tree corresponds to a partial packing of l circles

into C. The potential of each node of the tree is assessed via a look ahead strategy that, starting with the partial packing of the current node, assigns each unpacked circle to its maximum hole degree position. The beam search stops either when the look ahead strategy identifies a feasible packing or when it has reviewed all nodes.

Pintér and Kampas [2006] present numerical results obtained using Lipschitz Global Optimizer (LGO). Castillo Castillo et al. [2008] applies various off-the-shelf generic global optimization techniques, and compare their performance. They further improve the results of the generic solvers by implementing a posteriori strategy that, given a near-optimal initial arrangement, swaps all pairs of adjacent sized circles until no possible improvement exists.

Addis et al. [2008] present a strategy for optimally placing circles in a smallest circle. They mix standard local optimization routines with local moves between minima, while reinforcing solution dissimilarity but reducing the solution space. The resulting approach obtains the best known solution for problems of up to 50 circles and $r_i = i$, $i = 1, \ldots, N$.

Lu and Huang [2008] incorporate the principle of maximum cave degree for corner occupying actions into an improved "population control" (PERM) and propose a placement heuristic called A0 to quickly pack the circles into the container. Corner occupying strategy is an existing quasi-human method which has been successfully used in rectangle and circle packing problem. Subsequent PERM search strategy is presented and combined with A0 to solve the circle packing problem. This algorithm is less efficient than Zhang and Deng [2005] for several large-scale identical circle instances.

Hifi and M'Hallah [2009a] show that the combinatorial structure and the continuous optimization aspects of CPP should not be treated individually, but must be considered simultaneously. They base their recommendation on the comparison of the performance of two algorithms: Beam Search 1 (BS1) and Beam Search 2 (BS2). BS1 is a two-stage approach. The first stage uses a beam search to identify the best ordering of the circles. The second stage considers the circles in the order identified in stage 1, and uses a beam search to find the best position of each circle.

Al-Modahka et al. [2011] present an adaptive hybrid algorithm that addresses the combinatorial structure of CPP via a Tabu Search (TS), and its continuous optimization

aspects via a combination of nested partitioning (NP) and nonlinear optimization. The hybrid TS/NP algorithm exploits the advantages of TS to undertake a local search aimed at identifying a good permutation of the circles, whereas NP undertakes a global search to identify their respective best positions. The provided results are further modified/improved using some diversification strategies.

Akeb et al. [2011] address the circular open dimension problem (CODP) as in Stoyan and Yas'kov [2004] but they consider non-identical circles of fixed radii. In CODP the authors presents an initial strip of fixed width W and unlimited length, as well as a finite set N of n circular pieces C_i of known radius r_i $i \in N$. The objective is to search for a global optimum corresponding to the minimum length of the initial strip containing the n pieces. 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 combines the multi-start and separated beam strategies. They test their algorithm on two sets of instances: one set is composed of instances presented in the literature and the second set is composed of 1560 randomly generated instances where the number of circles n is taken from the following discrete interval $10, 20, 25, 30, 35, 40, 50, 60, 70, 80, 90, 120, 150$. According to their results their algorithm dominates those against which it was compared.

Carrabs et al. [2014] proposed an algorithm that by applying a strength along a selected direction on each circle, simulates the shifting of circles on the plane and tries to reduce the radius of the circular container during this movements. The algorithm is based on a multistart technique where the starting solutions are produced by a tabu search heuristic that uses also the current best solution.

2.3 Evolutionary Computation

ECPP (Evolutionary Computation for Circle Packing Problem) relies on the parameter values that encode solutions, which are initially randomly distributed between lower and upper bounds. Non-overlapping constraints are easily broken; it is not easy to avoid producing unfeasible solutions during the initialization process. The modifications of valid solutions through crossover and mutation in Genetic Algorithms for example, may produce unfeasible solutions as well as feasible.

George et al. [1995] considered the problem of fitting pipes of different diameters into a shipping container, here they study the subproblem of fitting circles of different sizes into a rectangle since that problem is a central part of the larger problem. They formulated the packing problem as a mixed-integer non-linear programming problem: continuos 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 maximize 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 variable. 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 implements heuristics it is important to obtain a stable solutions 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.

Zhi-Qin et al. [2001] propose a Human-Computer Interactive Genetic Algorithm for solving the two-dimensional constrained layout optimization problem. The algorithm composes chromosomes with artificial individuals (AIs) and divides the population into subgroups. Each subgroup, has different values of crossover and mutation probabilities. After copy, crossover, and mutation, the best individual in each subgroup is transferred to adjacent subgroups. New AIs are determined based on the value of the fitness function. These new AIs are copied in order to ensure they play an important role in chromosome population. Then they are placed into the chromosome population to replace the worse individuals. The steps mentioned above are repeated until the human expert finds a satisfactory solution.

Hifi and M'Hallah [2003] studied the problem of cutting a rectangular plate R of dimensions (L, W) into as many circular pieces as possible. The circular pieces are of n different types with radii r_i $i = 1, \ldots, n$. They solve the *constrained circular* problem, where d_i

the maximum demand for piece type i is specified, using two heuristics: a constructive procedure-based heuristic an a genetic-based heuristic. Both of these approaches search a good ordering of the pieces and use an adaptation of the *best local position* procedure to find the "best" layout of this ordered set. This positioning procedure is specifically tailored to circular cutting problems. It acts, for constrained problems,as one of the mutation ooperators of the genetic algorithm. They compare the performance of both proposed approaches to that of existing approximate and exact algorithms on several problem instances taken from the literature.The instances considered are from Stoyan and Yas'kov [1998], the solutions given from the two heuristics developed were compared against solutions in Hifi et al. [2004], Stoyan and Yas'kov [1998] and dominated those in Hifi et al. [2004], Stoyan and Yas'kov [1998].

Xu et al. [2007] present a novel order-based positioning technique for the layout optimization problem. A permutation $(1, 2, \ldots, N)$ can yield a layout by specifying the order in which circles are placed. As there exist $N!$ possible permutations for N circles, the GA is an appropriate technique to use in order to search such a large space. The GA is used to evolve the placement order of each circle.

Yan-jun et al. [2010] proposed an improved Evolution Strategy with Crossover Operator (ESCO) to tackle the constrained circle packing problem. The proposed ESCO extends a canonical ES to deal with combinatorial optimization by employing the crossover operator from genetic algorithms, aiming to exchange the location of circles for obtaining a better packing scheme. They aim to solve the general CPP; they measure the quality of the packing by the size of the container and the weighted average pair-wise distance between circles.

Yan-Jun et al. [2012] present a Layout Pattern-Based Particle Swarm Optimization algorithm (LPPSO) for solving the two-dimensional packing problem with constraints. It is assumed that there are two layout pattern S_1, S_2. If the relative position relationship of layout objects' centroid in S_1 is the same with that of S_2 (namely LP(S_1)=LP(S_2)), the layout pattern of the S_1 and S_2 is defined as the isomorphic, which is denoted by ILP(S_1,S_2). Conversely, if LP(S_1)$\neq$LP(S_2), the S_1 and S_2 are non-isomorphic, which is denoted by NLP(S_1,S_2). In the optimizing process of LPPSO, some individuals are constructed according to non-isomorphic layout patterns and these individuals are added into the current population of the Particle Swarm Optimization (PSO) algorithm to replace the worst individuals; the new population is created as a result. A non-isomorphic layout pattern is constructed based on an exact boundary line approach (the distance

between two circles) to avoid premature convergence and improve the computational efficiency.

Machado and Leitao [2011] propose an evolutionary algorithm that solves the packing problem of identical circles inside a square container, their approach is based on genetic algorithms. Each individual encodes a potential solution for the circle packing problem and a fitness function, which is used to assess the suitability of its potential mating partners. This approach relies on the observation that in some cases the optimal solution for the packing of n circles in a square may also encode optimal solutions for the packing of $n - i$ circles for i in $[1, n - 2]$. This observation made them consider the following hypothesis: the performance of an individual on a $n - i$ packing problem may provide information regarding its performance on the target n packing problem; this information may be valuable for mating selection purposes. 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 experimental results show that by evolving mating selection functions it is possible to surpass the results attained with hardcoded fitness functions. Moreover, they also indicate that genetic programming was able to discover mating selection functions that: use the information regarding potential mates in novel and unforeseen ways. The accuracy considered for the results was four decimal places.

To the best of the knowledge of the authors, there does not exist a satisfactory E.C. approach to solving CPP. There are not even previous publications available to compare our results with other approaches.

2.4 Final Remarks

In this chapter we gave a description of the two-dimensional packing problem. I presented a brief historical note highlighting appropriate examples of the two-dimensional packing problem. I 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 using Traditional Optimization and Evolutionary Computation.

Chapter 3

Population-based Solution

A population is a set of individuals where each individual represents a prospect solution to the problem. A Population-based solution spreads through a good search region, instead of only a good point, in the search space. In addition to representing a potentially good search region, the variance of the population members provides information about the extent of the potential search region. If new solutions are created in proportion to the variance of an existing population of points, a self-adaptive search procedure can be developed. In the start of such an algorithm with a randomly picked initial population of solutions, the variance in the population members is expected to be large, thereby ensuring a thorough exploration of the entire search space. On the other hand, during later iterations when the population of points have covered near the optimum, the variance in population members is expected to be small, thereby ensuring a focused search near the optimum. Without an external guidance, such a population-based algorithm can widen or narrow down its search power adaptively Deb [2004].

These algorithms are typically applied to hard problems with a large search space, where the presence of multiple solutions is exploited to find better search regions. The presence of multiple solutions in a search process allows diversity to be maintained and this can be beneficial in handling constrained optimization problems. A Population-based Solution usually yields the best values of the variables or the best scenarios which are an approximation to the optimum solution. A solution is considered optimum with respect to the best performance or best fitness, in terms of the objective function.

3.1 A General Evolutionary Computation Algorithm

Evolutionary Computation or Population-based algorithms are direct search methods, which use only objective function values to drive the search and employ more than one solution at each iteration. These Algorithms work on an encoding of the parameters set, search from a population of individuals, use an objective function, use probabilistic transition rules, and can provide a number of potential solutions to a given problem. Population-based algorithms allow direct generation of possible solutions in a single run.

Evolutionary computation is an ambitious name for a simple idea: use the theory of evolution as an algorithm. Any program that uses the fundamental structure shown in Algorithm 1 could be termed Evolutionary Computation. In an evolutionary algorithm, the first step is to create a population of individuals. The structures that describe those individuals are filled in at random. An objective function(fitness function for EAs) is used to decide which solutions deserve further attention. In the main loop of the algorithm, we pick solutions so that on average better solutions are chosen. This process is known as selection. The selected solutions are then subjected to variation. This variation can be in the form of random tweaks to a single structure or exchange of genetic material between structures. Changing a single structure is called unary variation or mutation. Exchanging material between structures is called an n-ary variation or crossover.

The main loop iterates the process of population updating via selection and variation. In accordance with the general theory of evolution, this should move the population toward fitter structures. This continues until you reach an optimum in the space of solutions, defined by your fitness function, or until a specified number of generations has been reached. This optimum may be the best possible place in the entire fitness space, or it may merely be better than all structures nearby in the search space. Adopting the language of optimization, I call these two possibilities global and local optima. Unlike many other types of optimizers, an evolutionary algorithm can jump from one optimum to another.

Algorithm 1 shows the basic structure of an evolutionary algorithm; in our case we add a step that tests the feasibility of each individual. To maintain our perspective of a high level of abstraction (i.e., working at the feasible space), after an individual is generated (steps 1 and 5) we apply a repairing process. This process repeats until computes the fitness of each individual (i.e., determines the radius of the enclosing circle) reaching an specified number of generations.

Algorithm 1 GenericEA

```
1: Generate a random population of individuals
2: repeat
3:    Test the individuals for quality
4:    Select individuals to be perturbed
5:    Produce new variations of selected individuals
6:    Replace old individuals with new ones
7: until termination criterion is satisfied
```

3.2 Repulsion-based Repair

A configuration $P = ((x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N))$ represents fixed positions for the N circles. The repairing process can be stated as finding coordinates (x_i, y_i) of every circle c_i so that the whole configuration does not violate the non-overlapping constraint.

In this section is propose an algorithm inspired on a physical analogy. We imagine the set of circles as elastic rings. Under this analogy, circles that are too close together (i.e. overlapping circles) starts deforming and exerting a restoring force that pushes them apart. We allow these forces to push circles to the point where they touch, which is the point where the overlap disappears. It is necessary calculate the Center of Gravity P_g of the given circles to sort them by proximity to this Center.

Given two circles c_i and c_j where $i \neq j$ in the given configuration. If $(x_i - x_j)^2 + (y_i - y_j)^2 < (r_i + r_j)^2$, then c_i and c_j overlap, then the overlapping depth d_{ij} between them is

$$d_{ij} = r_i + r_j - \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, (i, j) \in \{1, \ldots, N\}^2 \tag{3.1}$$

$$d_{ij} \begin{cases} < 0 & \textit{distant circles} \\ = 0 & \textit{tangent circles} \\ > 0 & \textit{overlapping circles} \end{cases} \tag{3.2}$$

Since the initial configuration is generated randomly, there most likely exist overlaps among the circles. According to the physical analogy, these circles are considered as elastic rings. If there exists overlap between objects, they must have elastic forces acting on them, so they will move with the action of those forces.

Figure 3.1, shows two circles c_i and c_j that overlap; c_j will move along the direction from i to j by the reaction of elastic forces. How far it moves depends on its embedding

depth d_{ij} (3.1), c_j moves away from c_i a distance d_{ij} until it does not overlap with c_i. According to the following equations the coordinates of c_j are modified.

$$\theta_{ij} = tq^{-1}\frac{x_j - x_i}{y_j - y_i} \tag{3.3}$$

$$dx_j = (r_i + r_j)\ cos(\theta_{ij}) \tag{3.4}$$

$$dy_j = (r_i + r_j)\ sin(\theta_{ij}) \tag{3.5}$$

where dx_j is the projection of d_{ij} in the horizontal axis x and dy_j is the projection of d_{ij} in the vertical axis y. Therefore, Figure 3.2 shows the correction that restores overlapping with new position P'_j as follows:

$$x'_j = x_j + dx_j \tag{3.6}$$

$$y'_j = y_j + dy_j \tag{3.7}$$

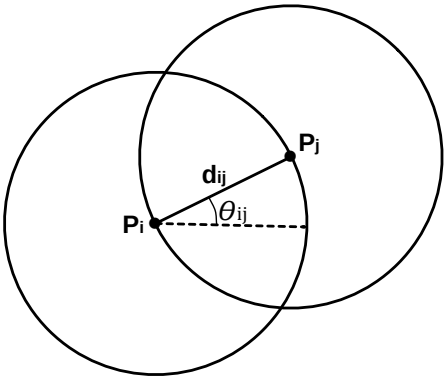


FIGURE 3.1: Two Overlapping Circles

The overlapping problem for two circles is solved by (3.1) − (3.7). If we have more than two circles, we need to define the order in which their positions are corrected, so that it guarantees that, at the end, no overlaps exist and that corrected circles are not moved in subsequent corrections (otherwise the algorithm could fall into an infinite cycle of corrections).

Algorithm 2 proposes such an order, guaranteeing both properties mentioned in the previous paragraph. Let us define P_g as the center of gravity of the set of circles. $P_g = \{\overline{x}, \overline{y}\}$, where $\overline{x}$ and $\overline{y}$ are the means of the coordinates of the Centers of all circles.

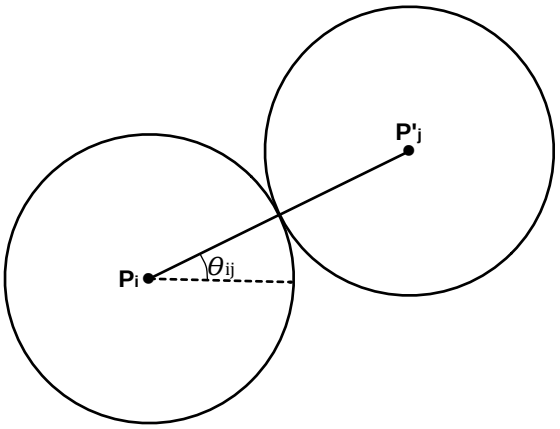


FIGURE 3.2: Two Repaired Circles

The main idea is to compute the center of gravity of the set of circles, and correct all possible overlaps from P_g, outwards.

Algorithm 2 RBRepair (C)

```
1: C: set of Circles
2: P_g = (x_g, y_g) = Center of Gravity of Circles
3: Sort C by proximity to P_g
4: for i = 1 to |C| − 1 do
5:    for j = i + 1 to |C| do
6:       if c_j overlaps c_i then
7:          c_j ← Repair c_j to avoid overlapping, according to (3.4) − (3.7)
8:       end if
9:    end for
10: end for
11: return C (without overlapping)
```

If the previous configuration is $P = ((x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N))$, then the new configuration is $P' = ((x_1, y_1), (x'_2, y'_2), \ldots, (x'_N, y'_N))$ represents the center coordinates of circles without overlapping. Note that c_1 is the closest circle to P_g, therefore it does not change with the execution of Algorithm 2.

It is easy to prove that the computational complexity of Algorithm 2 is $O(N^2)$. Since Algorithm 2 does not have the means to find the closest neighbors of each circle, it compares every circle with every other remaining one. Therefore, it is important to seek the implementation of repairs that are faster and consume less computational resources. An alternative approach is proposed in the next section.

3.3 Delaunay Triangulation-based Repair

A configuration P represents fixed positions of the centers of the N circles. The repairing process can be stated as finding coordinates P_i' of every circle c_i so constraint 1.2 is not violated.

The repairing process described in this subsection consists on removing all overlaps using the Delaunay Triangulation. The Delaunay Triangulation is a triangulation such that the circumcircle of each triangle does not contain another point in its interior.

The set $T = \{T_1, \ldots, T_{N_t}\}$ represents the Delaunay Triangulation formed by P. Each triangle $T_k = (P_{k_1}, P_{k_2}, P_{k_3})$ of T is a triplet of circle identifiers.

Consider a set of points P, in the Euclidean plane, where $P_i = (x_i, y_i)$ and $|P| \geq 3$. Assume that these points are not all collinear, and that no four points are cocircular. Let $d(P_i, P_j)$ denote the Euclidean distance between points P_i and P_j. A triangulation T of a set of N points P is considered a Delaunay Triangulation of P if either of the following lemmas hold Lee and Schachter [1980]:

Lemma 1. Given a set P of N points, any triangulation $T(P)$ has the same number of triangles,

$$N_t = 2(N-1) - N_h \tag{3.8}$$

and the same number of edges,

$$N_e = 3(N-1) - N_h \tag{3.9}$$

where N_h is the number of points on the Convex Hull of P.

Lemma 2. Given a set P of points, any *edge* (P_i, P_j) is a Delaunay edge of $DT(P)$ if and only if there exists a point x such that the circle centered at x and passing through P_i and P_j does not contain in its interior any other point of P.

Lemma 3. Given a set P of points, a triangle $T_k = (P_{k_1}, P_{k_2}, P_{k_3}) \in DT(P)$ if and only if its circumcircle does not contain any other point of P in its interior.

Among the possible triangulations of a set of points, Delaunay Triangulations are an interesting alternative as they tend to generate triangles that maximize the minimum

angle of all the triangles in the triangulation. Additionally, the performance of Delaunay triangulations is acceptable in practice; the algorithm has a complexity time of $O(N \log N)$. The repulsion-based repair algorithm does not use information about neighborhood of circle. The DT-based repulsion algorithm uses the neighborhood information provided by the DT to repair triplets of triangles at once, not having to verify overlaps with any other circle.

For CPP the set P of (3.10) is formed by the coordinates (x, y) of the centers of the circles.

$$P = \{P_1, \ldots, P_N\}, P_i = (x_i, y_i), i \in [1, 2, \ldots, N] \tag{3.10}$$

Constraint 3.11 represents the conditions to apply by the repair process based on the Delaunay Triangulation, i.e., the three circles after the repair process must be as close as possible to each other, and at last one of the three must be tangent with the other two.

$$\begin{aligned} \sqrt{(x_{k_1} - x_{k_3})^2 + (y_{k_1} - y_{k_3})^2} &= r_{k_1} + r_{k_3}, \\ \sqrt{(x_{k_2} - x_{k_3})^2 + (y_{k_2} - y_{k_3})^2} &= r_{k_2} + r_{k_3}, \\ k_1 \neq k_2 \neq k_3, k_1, k_2, k_3 &\in [1, 2, \ldots, N] \end{aligned} \tag{3.11}$$

The repair process starts with $k = 1$ for $T_k = (P_{k_1}, P_{k_2}, P_{k_3})$, the first triangle in T, where P_{k_i}, corresponds to circle c_{k_i} of T_k, for $1 \leq i \leq 3$. Circle c_{k_i} has coordinates P_{k_i}. The Delaunay Triangulation-based repair process first repairs P_{k_1} and P_{k_2} of T_k according to (3.1) − (3.7). We stablish that P_{k_1} will not be moved and P_{k_2} is attracted to or repelled from P_{k_1} to make their circles tangent. P'_{k_2} is the repaired position of P_{k_2}. Circle c_{k_2} is moved along the line that connects P_{k_1} with P_{k_2}, to a place where (3.12) is satisfied

$$\sqrt{(x_{k_1} - x'_{k_2})^2 + (y_{k_1} - y'_{k_2})^2} = r_{k_1} + r_{k_2} \tag{3.12}$$

The third circle c_{k_3} is attracted to or repelled from c_{k_1} and c'_{k_2}, its new coordinates x'_{k_3} and y'_{k_3} can be computed as follows:

$$\theta_{ijk} = cos^{-1} \frac{(r_{k_1} + r_{k_3})^2 + (r_{k_1} + r_{k_2})^2 - (r_{k_3} + r_{k_2})^2}{2(r_{k_1} + r_{k_3})(r_{k_1} + r_{k_2})} \tag{3.13}$$

$$x'_{k_3} = x_{k_1} + (r_{k_1} + r_{k_3}) cos(\theta_{ij} + \theta_{ijk}) \tag{3.14}$$

$$y'_{k_3} = y_{k_1} + (r_{k_1} + r_{k_3}) sin(\theta_{ij} + \theta_{ijk}) \tag{3.15}$$

I.e., given three circles c_{k_1}, c_{k_2}, and c_{k_3}, c_{k_2} is attracted to or repelled from c_{k_1}, x'_{k_2} and y'_{k_2} are calculated with (3.1) − (3.7). The third circle c_{k_3} is attracted to or repelled from c_{k_1} and c'_{k_2}, its new coordinates x'_{k_3} and y'_{k_3} are calculated according with (3.13) − (3.15). The remaining triangles will be repaired in an order such that every triangle being repaired is adjacent to a repaired triangle, according to the Delaunay Triangulation. Under those circumstances, two of the circles of T_k (the triangle under repair) have already been fixed and will not be moved again. Assume $T_k = (P_{k_1}, P_{k_2}, P_{k_3})$, where P_{k_1} and P_{k_2} have been fixed; that leaves the only option of moving the remaining P_{k_3}, to repair triangle T_k. P_{k_3} will be moved to a place according to (3.13) − (3.15) where it satisfies (3.16).

$$\begin{aligned} \sqrt{(x'_{k_1} - x'_{k_3})^2 + (y'_{k_1} - y'_{k_3})^2} &= r_{k_1} + r_{k_3}, \\ \sqrt{(x'_{k_2} - x'_{k_3})^2 + (y'_{k_2} - y'_{k_3})^2} &= r_{k_2} + r_{k_3}. \\ k &\in [2, 3, \ldots, N_t] \end{aligned} \tag{3.16}$$

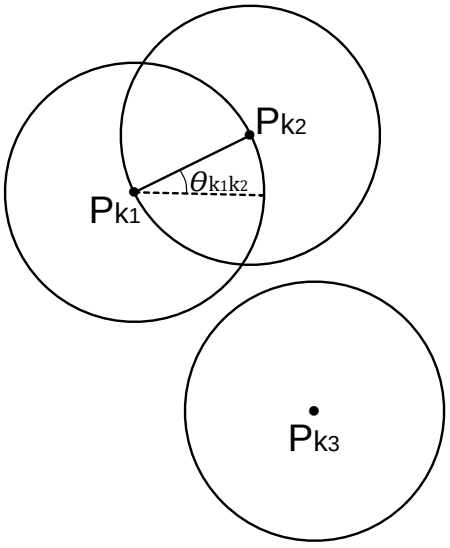


FIGURE 3.3: DT-Initial Configuration

The order in which circles are repaired is important. We need to produce an efficient algorithm that repairs every circle only once, and once it is fixed it will not be moved again. On the other hand, if the order is arbitrary, the repair process can get to a point where overlaps exist and no circles can be moved to solve the situation. To avoid these problems and guarantee convergence of the repair process we compute the Center of Gravity for the set of circles, then sort them by ascending distance to the Center of Gravity. Once we have the circles sorted, we compute the Delaunay Triangulation, which yields the neighbors of each circle in triplets. We will repair these triplets attracting

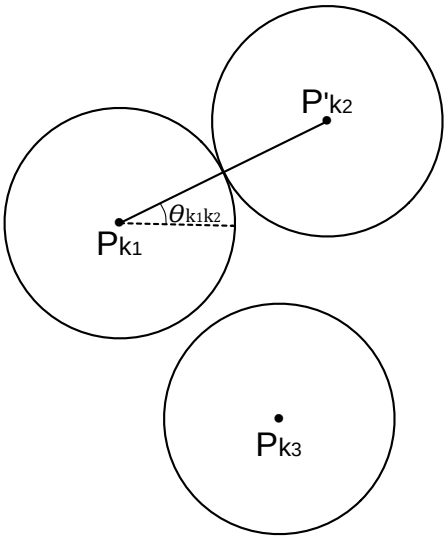


FIGURE 3.4: DT-Two repaired circles

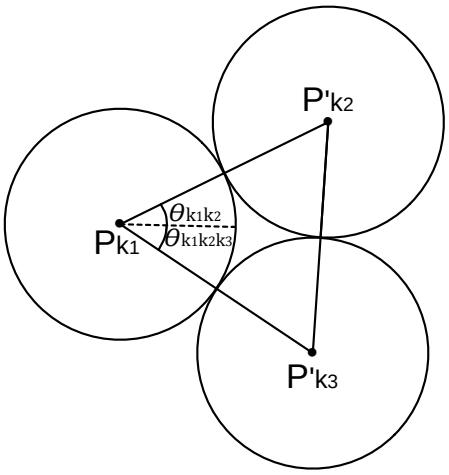


FIGURE 3.5: DT-Three repaired circles

circles to the Center of Gravity. Algorithm 3, DTRepair, implements this process. DTRepair takes as input a set of circles and returns a set of repaired circles.

3.4 Enclosing Circular Container

Once a configuration contains no overlaps, we need to determine the size of the smallest circle that contains all circles in the configuration. Enclosing finds the coordinates $\{x_0, y_0\}$ that minimize r_0, the radius of the containing circle c_0.

Algorithm 3 DTRepair (C)

1: C : set of $Circles$
2: $P_g = (x_g, y_g)$ = Center of Gravity of C
3: Sort C by ascending distance to P_g
4: $T \leftarrow$ Delaunay Triangulation of the centers in $Sorted\ C$
5: $Coordinates$ of $C(T_1) \leftarrow Repair\ C(T_1)$, according to (3.1) − (3.15)
6: **for** $k = 2$ to $|T|$ **do**
7: $\quad Coordinates$ of $C(T_k) \leftarrow Repair\ C(T_k)$, according to (3.13) − (3.15)
8: **end for**
9: **return** C (without overlapping)

To compute the enclosing circle, Algorithm 4 sorts the circles by descending distance from their center of gravity. It takes the farthest circle as the starting enclosing circle, and takes each circle at a time, checking whether or not it is contained in the previous enclosing circle. If it is not, it computes the new enclosing circle.

Assume at a given iteration i, circle c_i, has center coordinates (x_i, y_i), and container circle c_0, has coordinates (x_0, y_0). If $\sqrt{(x_0 - x_i)^2 + (y_0 - y_i)^2} + r_i < r_0$, then c_0 completely contains c_i, and there is nothing to do. Otherwise, we need to compute a new c_0 that encloses all of them. In that case, the radius of the new enclosing circle r'_0 is

$$r'_0 = \frac{\sqrt{(x_0 - x_i)^2 + (y_0 - y_i)^2} + r_0 + r_i}{2} \tag{3.17}$$

Once we know the new radius, we compute the changes in the x and y coordinates of c_0's center.

$$\theta_{0i} = tg^{-1}\frac{x_0 - x_i}{y_0 - y_i} \tag{3.18}$$

$$dx_0 = (r'_0 - r_0)\ cos(\theta_{0i}) \tag{3.19}$$

$$dy_0 = (r'_0 - r_0)\ sin(\theta_{0i}) \tag{3.20}$$

where dx_0 and dy_0 are the changes to c_0's coordinates. Finally, the new position of c_0 is given by (3.21) and (3.22).

$$x'_0 = x_0 + dx_0 \tag{3.21}$$

$$y'_0 = y_0 + dy_0 \tag{3.22}$$

Note that as Algorithm 4 proceeds towards the center of gravity, the likelihood to modifying the container circle decreases. When have revised that all circles are inside of the enclosing circle c_0, the process has finished and the Container Circle c_0 is returned.

Algorithm 4 Enclosing (C)

1: C : set of $Circles$
2: $P_g = (x_g, y_g)$ = Center of Gravity of C
3: *Sort* C by descending distance to P_g
4: $c_0 \leftarrow c_1$
5: **for** $i = 2$ to $|C|$ **do**
6: **if** c_0 *does not enclose* c_i **then**
7: *Generate* a new *Container* c_0, according to (3.17) − (3.22)
8: **end if**
9: **end for**
10: **return** c_0

3.5 Time Complexity

It is well known that sorting and computing the Delaunay Triangulation takes time $O(N\ log\ N)$ Lee and Schachter [1980]. The loop in lines $6-8$ of Algorithm 3 repeats $O(N_t)$ times, and repair can be done in constant time, and since $N_t = O(N)$. The total times complexity of Algorithm 3 is $O(N\ log\ N)$.

Enclosing also sorts the sets of circles, taking $O(N\ log\ N)$ time. The loop in lines $5-9$ of Algorithm 4 repeats $O(N)$ times, and verification of enclosure and modifying the container can be done in constant time.

The total time complexity of Enclosing is $O(N\ log\ N)$. In conclusion, given the chromosome of an individual in the evolutionary process, repairing it and measuring its fitness (i.e. determining the enclosing circle) can be done in $O(N\ log\ N)$ time.

3.6 Final Remarks

This chapter introduced some basic concepts of Population-based Solutions. These concepts include evolutionary algorithms, population of individuals, fitness space, objective function, selection process, and feasibility of each individual and repairing process, which ones are used to obtain the enclosing container.

This chapter proposed two repair mechanisms called Repulsion-based repair and Delaunay triangulation-based repair, to apply to those unfeasible individuals to make them

feasible and proceed to enclosing process. Finally as we can see Delaunay triangulation-based repair process has a less time complexity less than Repulsion-based repair process, it means that consumes less time to obtain a feasible solution.

Chapter 4

Results

ECPP was tested under different conditions, and the results were compared to known solutions (either exact or approximate) known in the field. The problem of packing circles in a container has two known variants, the case where all circle sizes are equal (known as unit circle packing), and the case where the circles have different sizes as in Castillo et al. [2008]. ECPP was tested following this distinction.

Those tests are reported on the following two subsections. ECPP was implemented in Mathematica. I ran our experiments using a Mac computer with an Intel Core i7, 2 GHz, four cores, and 4 GB of RAM.

4.1 Packing Circles into a Circular Container

The first set of experiments tests ECPP's performance on unit-circle problems. These tests were conducted in two orthogonal directions; I compared the performance of two metaheuristic search algorithms (namely, Genetic Algorithms GA, Evolution Strategies ES, Differential Evolution DE, and Particle Swarm Optimization PSO) using the repulsion-based and the Delaunay Triangulation repair heuristics. These results can be compared against the proven optimal analytical solutions.

The second set of experiments tests ECPP's performance on uneven circle problems. Since there are no known optimal results for these problems, I performed two experiments to test ECPP's performance: the first one compares our results with those published at Mathrec Zimmermann [2006], the second one analyses ECPP's performance for

different values of variance of the radii of the circles. I analyzed the algorithm's performance with respect to the solutions' packing density; when the variance of the circles' radii in a problem instance increases, large circles produce holes that can be occupied by small circles, thus increasing the density of the produced solution.

4.1.1 Unit Circle Tests

To measure the performance of the two repair mechanisms I compared the numerical performance of four Metaheuristics (Genetic Algorithms, Evolutionary Strategies, Differential Evolution, and Particle Swarm Optimization) applied to the Circle Packing Problem. The most recent solutions to the circle packing problem are reported at the website Packomania Specht [1999]. This website started in 1999 reporting proven optimal solutions from 1 to 1500 unit circles and some cases of uneven circles. Unfortunately they do not provide computational time. To illustrate the effectiveness of our approaches, I tested problems of different sizes, from 3 circles to 100 circles. Only some of them are reported for the sake of brevity. I compared the results produced by our algorithms with the benchmarks presented in the Packomania web site Specht [1999].

The metrics used in the comparison of results are radius and density. Radius is the Best-known solution proven mathematically for a particular problem and density describes the ratio of total area occupied by the circles to container area, i.e. the relationship between the area of the container circle and the sum of areas of the circles that are inside it. Benchmarks are registered on tables 4.1 and 4.2 in the column labeled *Optimal*, with their corresponding radii and densities.

The results of the two repair mechanisms (Repulsion-based repair and DT-based repair) are presented in sections 4.1.1.1 and 4.1.1.2, respectively. Tables 4.1 and 4.2 compare our approaches with the benchmarks. The results are shown for the size of the instances, the radius of the container circle and the density corresponding to the container circle of the mean solution obtained with Genetic Algorithms (GA) and Differential Evolution (DE).

4.1.1.1 Results Using Repulsion-based Repair

Table 4.1 is composed of twelve columns: the first column refers to the number of unit circles to be packed; the second and third columns refer to the radius and density, respectively, of the optimal solution; the fourth and fifth columns show the mean radius and density obtained applying GA, the sixth and seventh columns show the mean radius and density obtained applying DE, the eighth and ninth columns show the mean radius and density obtained applying ES, the tenth and eleventh columns show the mean radius and density obtained applying PSO, the twelfth column shows the mean time required. The results produced by our approach were obtained by 30 independent executions with random initial populations. The table shows the mean of the 30 executions. The values highlighted in each row represent the best mean obtained.

N	*Optimal*		*GA*		*DE*		*ES*		*PSO*		Time(sec)
	r_0	*density*	r_0	*density*	r_0	*density*	r_0	*density*	r_0	*density*	
2	2.000000	0.500000	**2.000000**	**0.500000**	**2.000000**	**0.500000**	**2.000000**	**0.500000**	**2.000000**	**0.500000**	34
3	2.154700	0.646171	2.159630	0.643222	**2.159420**	**0.643347**	2.167570	0.638521	2.162410	0.641574	792
4	2.414213	0.686294	2.459890	0.661043	**2.415430**	**0.685601**	2.417340	0.684515	2.542500	0.618782	1760
5	2.701301	0.685211	2.949650	0.574686	**2.831360**	**0.623705**	2.839830	0.619993	2.954790	0.572686	2528
6	3.000000	0.666667	3.076380	0.633975	**3.000000**	**0.666667**	3.002450	0.665581	3.082950	0.631275	4240
7	3.000000	0.777778	3.265700	0.656367	**3.108350**	**0.724498**	3.137180	0.711243	3.245330	0.664630	5472
8	3.304764	0.732504	3.543870	0.636993	**3.430100**	**0.679950**	3.448400	0.672751	3.519480	0.645853	11024
9	3.613125	0.689406	3.872370	0.600190	**3.777500**	**0.630714**	3.780407	0.629746	3.836300	0.611531	14720
10	3.813025	0.687796	4.089480	0.597948	**3.941590**	**0.643660**	3.952120	0.640234	4.074380	0.602389	14816
11	3.923804	0.714462	4.218920	0.618003	**4.132690**	**0.644062**	4.135111	0.646959	4.267180	0.604103	14940
12	4.029601	0.739022	4.434350	0.610269	**4.285840**	**0.653294**	4.290090	0.653697	4.467100	0.601354	14976
13	4.236067	0.724464	4.614080	0.610622	**4.433790**	**0.661291**	4.444030	0.658249	4.689200	0.591215	15068
14	4.328428	0.747252	4.837260	0.598314	**4.678800**	**0.639528**	4.682396	0.638545	4.886960	0.586207	15276
15	4.521356	0.733758	4.992760	0.601740	**4.900330**	**0.624656**	4.925540	0.618278	4.997750	0.600541	15400
16	4.615425	0.751096	5.185520	0.595024	**5.037030**	**0.630624**	5.037680	0.630463	5.203530	0.590914	15408
17	4.792033	0.740304	5.316410	0.601468	**5.193660**	**0.630235**	5.198960	0.628949	5.312000	0.602467	15520
18	4.863703	0.760920	5.491150	0.596961	**5.293990**	**0.642254**	5.301780	0.640368	5.490070	0.597195	15696
19	4.863703	0.803193	5.586120	0.608883	**5.432230**	**0.643869**	5.487330	0.631003	5.656340	0.593857	15972
20	5.122320	0.762249	5.738290	0.607387	**5.591260**	**0.639749**	5.603460	0.636968	5.772560	0.600196	16596
25	5.752824	0.755408	6.385920	0.613046	**6.312330**	**0.627423**	6.316960	0.626504	6.511200	0.589682	17516
30	6.197741	0.781016	6.998590	0.612491	**6.959850**	**0.619328**	6.999050	0.612411	7.111120	0.593260	18468
35	6.697170	0.780343	7.621730	0.602505	**7.483660**	**0.624942**	7.499360	0.622329	7.658540	0.596728	19128
40	7.123850	0.788190	8.126000	0.605768	**8.000540**	**0.624916**	8.078650	0.612890	8.158680	0.600925	20728
45	7.572910	0.784669	8.559630	0.614189	**8.478760**	**0.625962**	8.483370	0.625282	8.60686	0.607468	23992
50	7.947515	0.791604	9.011700	0.615682	**8.948360**	**0.624430**	8.971140	0.621262	9.104400	0.603208	29264
55	8.211100	0.815755	9.440760	0.617090	**9.333180**	**0.631399**	9.366770	0.626878	9.497040	0.609798	55460
60	8.646220	0.802599	9.870930	0.615794	**9.865880**	**0.616425**	9.914510	0.610392	9.945400	0.606606	78848
65	9.017400	0.799376	10.342000	0.607721	**10.179400**	**0.627294**	10.213100	0.623153	10.352700	0.606468	98532
70	9.345650	0.801454	10.665700	0.615343	**10.492600**	**0.635815**	10.502700	0.634599	10.677000	0.614047	117804
75	9.672029	0.801728	10.943500	0.626257	**10.939800**	**0.626674**	11.018200	0.617786	11.069200	0.612111	122044
80	9.968150	0.805120	11.352900	0.620693	**11.274100**	**0.629403**	11.318400	0.624484	11.378100	0.617949	128716
85	10.163100	0.822935	11.683800	0.622654	**11.543200**	**0.637923**	11.723100	0.618487	11.695800	0.621378	139332
90	10.546100	0.809210	11.992200	0.625815	**11.953100**	**0.629918**	12.095300	0.615193	12.018200	0.623110	142992
95	10.840200	0.808442	12.244100	0.633680	**12.195400**	**0.638755**	12.221400	0.636033	12.308300	0.627085	157080
100	11.082149	0.814241	12.612100	0.628675	**12.506500**	**0.639337**	12.814400	0.608985	12.768300	0.613381	190244

TABLE 4.1: Performance using Repulsion based Repair applying GA, DE, ES and PSO

From table 4.1 we can see that DE outperforms the other metaheuristic. This result was at all expected; it is vox populi in the Evolutionary Computation community, that DE is one of the best metaheuristics for real-valued optimization problems. Table 4.1 also shows, in the last column, the computational time required to solve each problem size. Since both GA and DE were run with the same population size and number of generations, their computational times are basically the same.

Even though the mean results do not seem to show clear superiority of GA in most cases, I decided to perform a statistical comparison of the result for selected cases. For size 30, I compared the performance of GA and DE, for a statistical difference in means, for a significance level $\alpha = 0.05$, proving that indeed GA's performance is superior to DE. For $n = 55$, I performed the same test, resulting in DE's performance proving superior to GA.

Figure 4.1 illustrates the plot formed by the mean values obtained for the solution of the four metaheuristics and the benchmarks corresponding for each problem size (i.e. number of circles). Figure 4.2 illustrates the plot formed by the Density obtained by the mean values of the fitness function for the four metaheuristics and the benchmarks corresponding for each problem size (i.e. number of circles). From Figures 4.1 and 4.2 we can see that the best densities were obtained by DE for most cases. Those figures show the distance of ECPP's solutions with respect to the optimal ones.

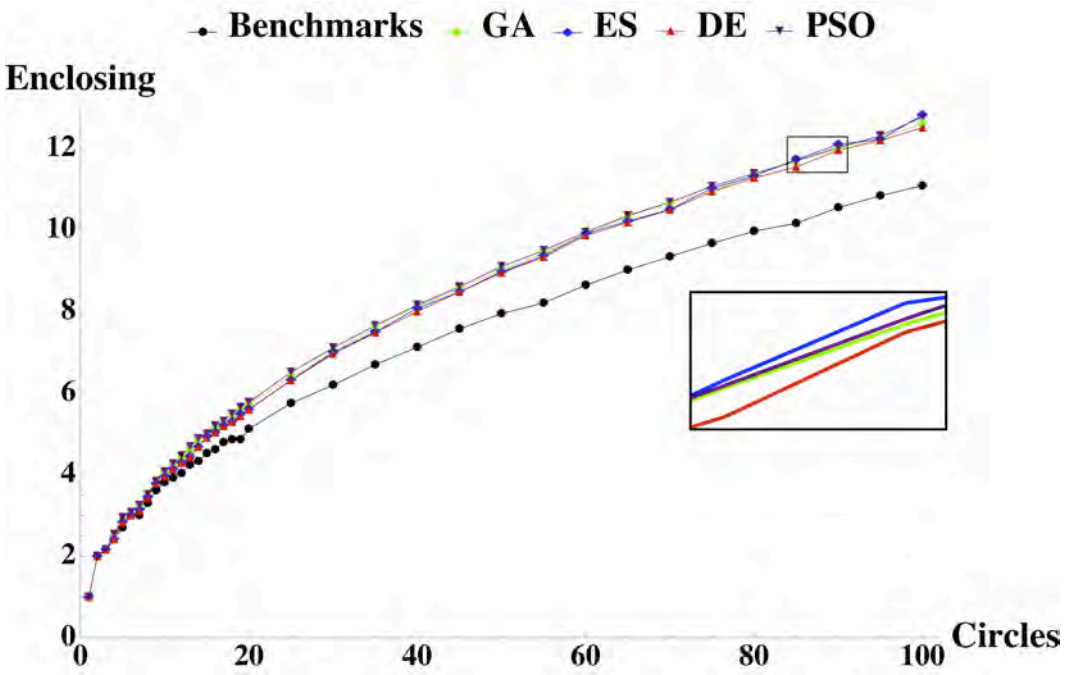


FIGURE 4.1: Radius of the Enclosing Circle using Repulsion-based Repair

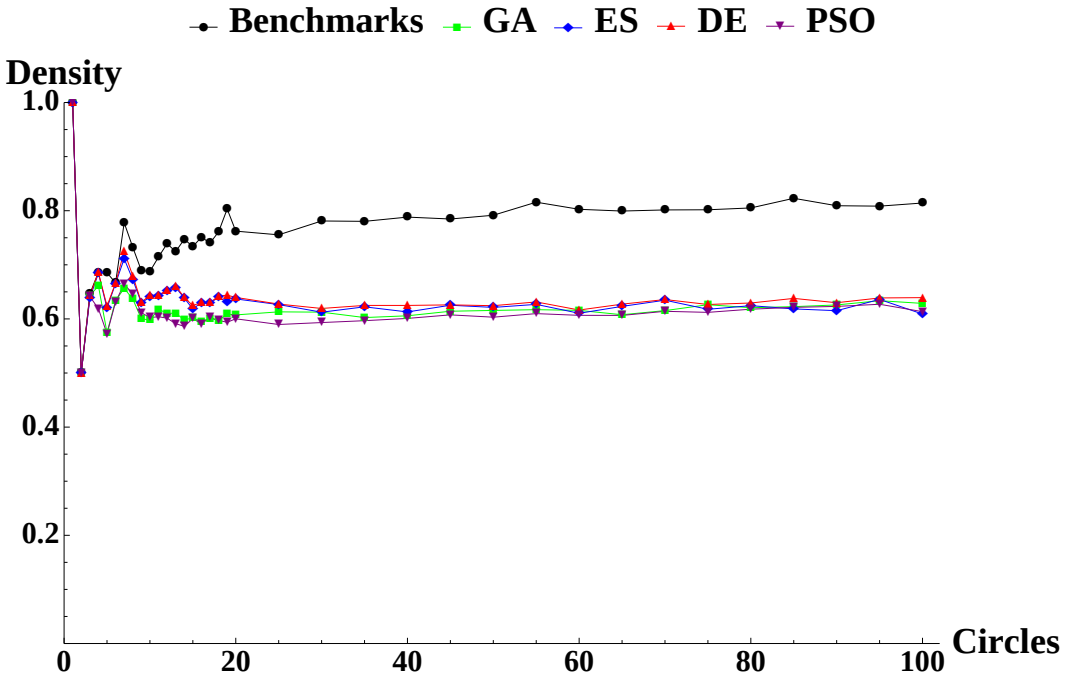


FIGURE 4.2: Density of the Enclosing Circle using Repulsion-based Repair

4.1.1.2 Results Using DT-based Repair

Table 4.2 has the same structure as Table 4.1. The results produced by our approach were obtained by 30 independent executions with random initial solutions. The table shows the mean of the 30 executions. The values highlighted in each row represent the best result obtained.

N	*Optimal*		*GA*		*DE*		*ES*		*PSO*		Time(sec)
	r_0	*density*	r_0	*density*	r_0	*density*	r_0	*density*	r_0	*density*	
2	2.000000	0.500000	**2.000000**	**0.500000**	**2.000000**	**0.500000**	**2.000000**	**0.500000**	**2.000000**	**0.500000**	34
3	2.154700	0.646170	**2.154700**	**0.646171**	**2.154700**	**0.646171**	**2.154700**	**0.646171**	**2.154700**	**0.646171**	204
4	2.414213	0.686291	**2.732050**	**0.535898**	**2.732050**	**0.535898**	**2.732050**	**0.535898**	**2.732050**	**0.535898**	408
5	2.701301	0.685210	**3.000000**	**0.555556**	**3.000000**	**0.555556**	**3.000000**	**0.555556**	**3.000000**	**0.555556**	680
6	3.000000	0.666666	**3.000000**	**0.666667**	**3.000000**	**0.666667**	**3.000000**	**0.666667**	**3.000000**	**0.666667**	1054
7	3.000000	0.777777	**3.000000**	**0.777778**	**3.000000**	**0.777778**	**3.000000**	**0.777778**	**3.000000**	**0.777778**	1190
8	3.304764	0.732502	**3.645750**	**0.601888**	**3.645750**	**0.601888**	**3.645750**	**0.601888**	**3.645750**	**0.601888**	2380
9	3.613125	0.689407	**3.800000**	**0.623269**	**3.800000**	**0.623269**	**3.800000**	**0.623269**	**3.800000**	**0.623269**	2958
10	3.813025	0.687797	**4.000000**	**0.625000**	**4.000000**	**0.625000**	**4.000000**	**0.625000**	**4.000000**	**0.625000**	3162
11	3.923804	0.714460	**4.055050**	**0.668960**	**4.055050**	**0.668960**	**4.055050**	**0.668960**	**4.055050**	**0.668960**	3808
12	4.029601	0.739021	**4.055050**	**0.729775**	**4.055050**	**0.729775**	**4.055050**	**0.729775**	**4.055050**	**0.729775**	4216
13	4.236067	0.724465	**4.550800**	**0.627724**	**4.573610**	**0.621477**	**4.550800**	**0.627724**	4.573610	0.621477	4556
14	4.328428	0.747252	**4.605550**	**0.660032**	**4.605550**	**0.660032**	**4.605550**	**0.660032**	**4.605550**	**0.660032**	4896
15	4.521356	0.733759	**4.752780**	**0.664043**	**4.752780**	**0.664043**	**4.752780**	**0.664043**	**4.752780**	**0.664043**	5508
16	4.615425	0.751097	**4.815756**	**0.689908**	**4.815760**	**0.689908**	**4.815756**	**0.689908**	**4.815760**	**0.689908**	5984
17	4.792033	0.740302	**5.000000**	**0.680000**	**5.000000**	**0.680000**	**5.000000**	**0.680000**	**5.000000**	**0.680000**	6290
18	4.863703	0.760918	**5.000000**	**0.720000**	**5.000000**	**0.720000**	**5.000000**	**0.720000**	**5.000000**	**0.720000**	6766
19	4.863703	0.803192	**5.187680**	**0.706005**	**5.187680**	**0.706005**	**5.187680**	**0.706005**	5.438890	0.642294	7480
20	5.122320	0.762248	5.467710	0.668990	**5.455490**	**0.671990**	**5.526910**	**0.654735**	5.506910	0.659499	7786
25	5.752824	0.755401	6.023700	0.688991	**6.003700**	**0.693588**	**6.003700**	**0.693588**	6.032980	0.686874	9860
30	6.197741	0.781006	6.465700	0.717612	**6.434870**	**0.724505**	6.523510	0.704950	6.626660	0.683175	10302
35	6.697170	0.780343	7.206830	0.673875	**7.056770**	**0.702840**	7.211470	0.673008	7.299210	0.656927	10472
40	7.123850	0.788190	**7.599570**	**0.692600**	7.600690	0.692396	7.686300	0.677058	7.735800	0.668420	12920
45	7.572910	0.784669	**8.000000**	**0.703125**	8.062160	0.692325	8.154820	0.676680	8.287090	0.655253	14212
50	7.947515	0.791602	8.550590	0.683877	**8.453270**	**0.699715**	8.521730	0.688517	8.685900	0.662736	16796
55	8.211100	0.815755	**8.825820**	**0.706078**	8.899030	0.694509	9.044070	0.672411	9.108500	0.662932	17850
60	8.646220	0.802599	**9.120590**	**0.721282**	9.264870	0.698992	9.422540	0.675796	9.474320	0.668429	21590
65	9.017400	0.799376	**9.568860**	**0.709893**	9.728280	0.686817	9.918440	0.660734	10.077000	0.640109	21760
70	9.345650	0.801454	**9.888930**	**0.715813**	10.051700	0.692814	10.40970	0.645986	10.408600	0.646126	23834
75	9.672029	0.801726	**10.379800**	**0.696120**	10.613000	0.665866	10.674400	0.658227	10.789400	0.644266	26690
80	9.968150	0.805120	**10.718000**	**0.696408**	10.861600	0.678115	11.046200	0.655644	11.122500	0.646679	28798
85	10.163100	0.822935	**11.057300**	**0.695221**	11.113700	0.688183	11.503100	0.642381	11.427600	0.650886	30396
90	10.546100	0.809210	**11.574400**	**0.671812**	11.582900	0.670823	11.743700	0.652583	11.750100	0.651864	32708
95	10.840200	0.808442	**11.748100**	**0.688311**	11.927700	0.667744	12.299200	0.628011	12.263400	0.631689	33626
100	11.082149	0.814239	**12.067400**	**0.686704**	12.433700	0.646843	12.617100	0.628180	12.650100	0.624904	36176

TABLE 4.2: Performance using Delaunay Triangulation based Repair applying GA, DE, ES and PSO

From table 4.2 we can see that GA outperforms the other metaheuristics. This result was not at all expected; it is vox populi in the Evolutionary Computation community, that DE is one of the best metaheuristics for real-valued optimization problems. The best metaheuristic was GA. As of today, the reason remains unknown to the authors of this work.

Figure 4.3 illustrates the plot formed by the mean values obtained for the solution for the four metaheuristics and the benchmarks corresponding for each problem size (i.e. number of circles). From Figure 4.3 we can see that the best solutions were obtained by GA.

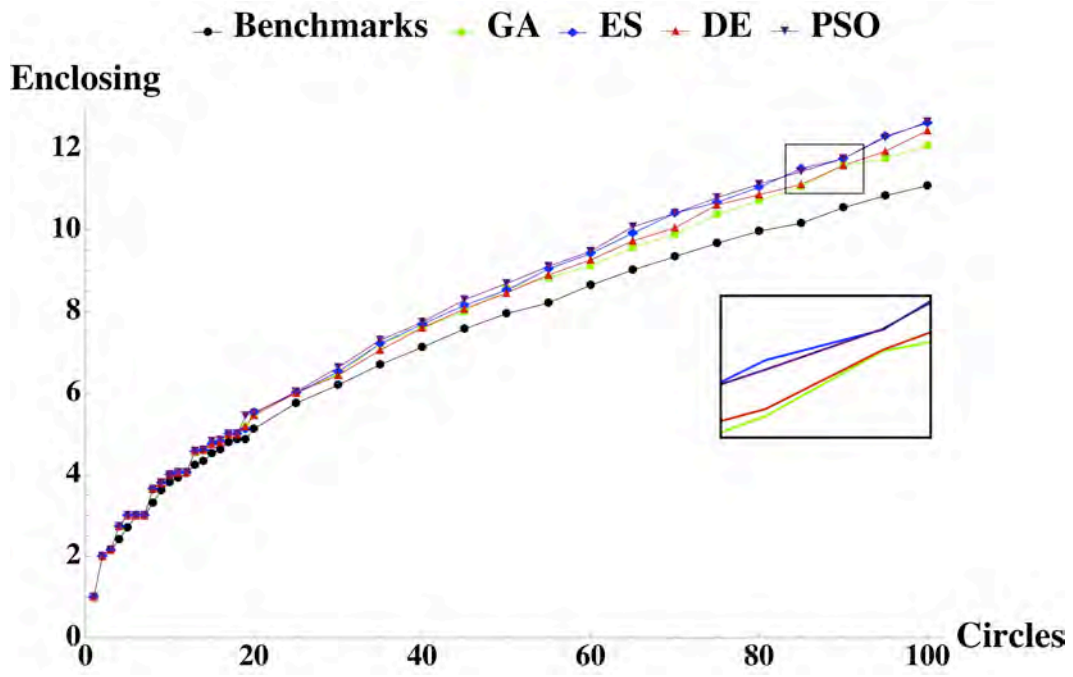


FIGURE 4.3: Radius of the Enclosing Circle using Delaunay Triangulation-based Repair

Figure 4.4 illustrates the plot formed by the Density obtained by the mean values of the fitness function for the four metaheuristics and the benchmarks corresponding for each problem size (i.e. number of circles). From Figure 4.4 we can see that the best densities were obtained by GA.

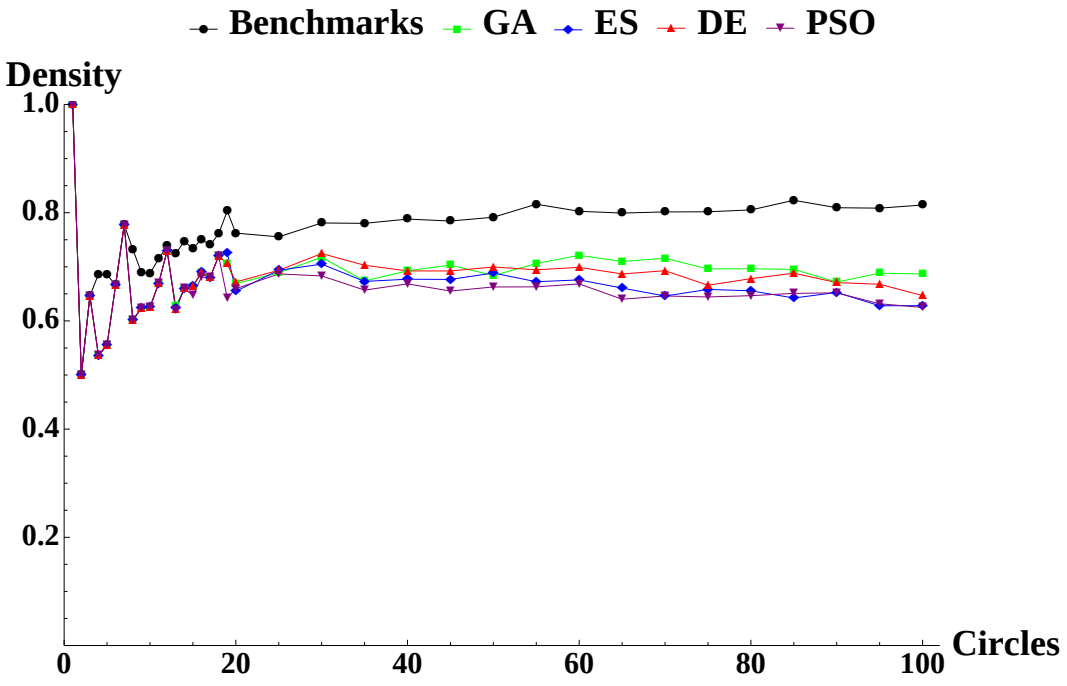


FIGURE 4.4: Density of the Enclosing Circle using Delaunay Triangulation-based Repair

4.1.2 Non-Unit Circle Tests

Although the unit circle version of CPP has been solved analytically for as many as 1014 circles Specht [1999], there is an infinite number of instance problems for each number of circles in the non-unit circle case, and there is no analytical solution for any of these problems.

In the absence of analytical solutions, in order to assess the performance of ECPP with non-unit circle problem instances, I performed two sets of experiments. First I compared ECPP's performance on the benchmark problems presented in Zimmermann [2006]. The problem presented in those web pages is to solve CPP for a set of circles where their radii $r_i = i$, $i = 1, \ldots, N$. Fig. 4.5 shows a plot of the radii of the enclosing circles of solutions obtained using Delaunay Triangulation-based Repair with Genetic Algorithms for problem sizes from 1 to 50. GA was executed with a population of 50 individuals and 500 generations.

The interpretation of these results have to take into account the following facts. The plot labeled as MathRec are the world records for these problems; no single algorithm provides the best solution to all problem instances. Even more, not all participants

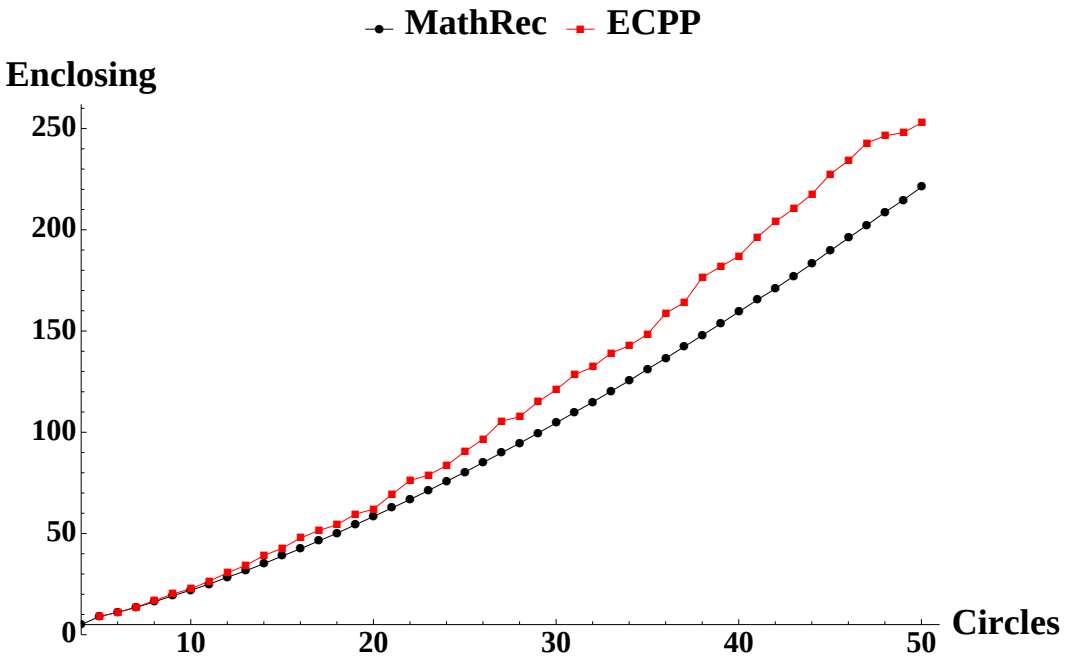


FIGURE 4.5: Performance Comparison: ECPP vs MathRec

provided solutions to all problem instances. In general algorithms are best at a single solution or at most at a range of them. None of the solutions presented in the MathRec contest was based on Evolutionary Computation. ECPP is the first attempt to solve CPP using metaheuristics of these kind.

In the second experiment to assess ECPP's performance on non-unit CPP we control the variance of the circle sizes and observe ECPPs performance (based on packing density) as the relative sizes of the circles vary. These experiments were conducted using only the best metaheuristic search and repair mechanisms found in the previous tests. That is, Genetic Algorithms, using the Delaunay Triangulation repair. For the sake of brevity, I only present experiments conducted to a set of 20 circles; which represent a considerably large search space of 40 dimensions.

I allow circle sizes to diverge from unity in such a way that I have control over the variance of their sizes. If the variance is zero, I have the unit-circle version of the problem, which was tested extensively and the results were reported in the previous section. As the variance increases, intuition leads us to suspect that smaller circles way fill in the gaps between larger circles, therefore increasing the density of the solutions.

For each problem size (i.e., number of circles), I generate the circle sizes according to the following probability distribution function:

$$U[1-\Delta, 1+\Delta] \tag{4.1}$$

where Δ represents the allowed amount of variation is well known that the variance of this distribution is

$$\sigma^2 = \frac{1}{12}((1+\Delta)-(1-\Delta))^2 = \frac{\Delta^2}{3} \tag{4.2}$$

so, by varying Δ we are controlling the variance of the circle sizes of the randomly generated problem instances.

I generated random problem instances for 20 circles varying Δ in the interval from 0 to 1, increasing delta by 0.1 for each experiment. For each setup, I performed 30 independent executions. This number of executions allow us to achieve statistical stability and draw conclusions about ECPP's performance on this type of problems. Fig. 4.6 shows a plot of the density of the solutions as Δ varies, for the described experiments.

4.1.3 Discussion

Comparing the four meta-heuristics using Delaunay Triangulation-based Repair, we can see that Genetic Algorithms performs best, followed closely by Differential Evolution, Evolutionary Strategies being further back, and, in last place, Particle Swarm Optimization.

As we can see in Figure 4.3 the difference between Genetic Algorithms and Differential Evolution is minimal and perhaps one might think that these results do not show obvious differences. However, statistical tests were performed for 55 to 100 circles, and there are indeed statistically significant differences between their respective means. For example for n = 55, we conducted a hypothesis test with a level of significance of 95 *percent*, where the null hypothesis was H_0:$\mu_{GA} = \mu_{DE}$ and the alternative hypothesis is H_1:$\mu_{GA} \neq \mu_{DE}$. The calculated value was $p = 0.074016$, which indicates that there is indeed a difference. Therefore, if there is a statistically significant difference between Genetic Algorithms and Differential Evolution which were the closest ones, there is also a

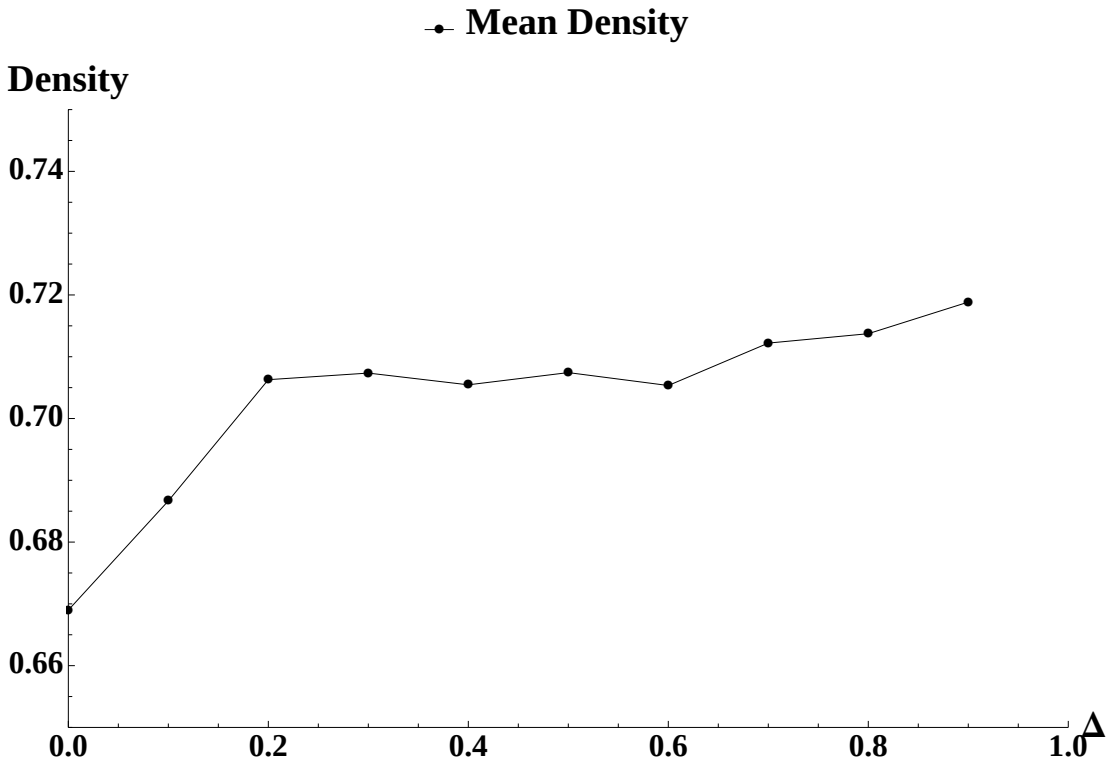


FIGURE 4.6: Mean Density of the Enclosing Circle with Uneven Circles using Delaunay Triangulation-based Repair

statistically significant difference between Genetic Algorithms and Evolutionary Strategies or Particle Swarm Optimization, whose means are more distant.

To date, we do not have explanation to why Genetic Algorithms outperforms Differential Evolution when applying the Delaunay Triangulation-based repair, even though most of the published literature mentions that Differential Evolution performs better than Genetic Algorithms in continuous optimization problems.

To compare the results obtained by the Repulsion and Delaunay Triangulation based repair processes we plot the mean of size of the enclosing circle for the different problem sizes. Figure 4.7 shows that the Delaunay Triangulation repair performs better than the Repulsion-based repair. Also, the repairing process by Delaunay Triangulation requires less time; that means less evaluations are needed to obtain a result closer to the optimum, than by the repulsion heuristic.

In very few cases, where the optimal configuration is one that can not be obtained by the Delaunay Triangulation-based repair, this repair method has trouble approaching the

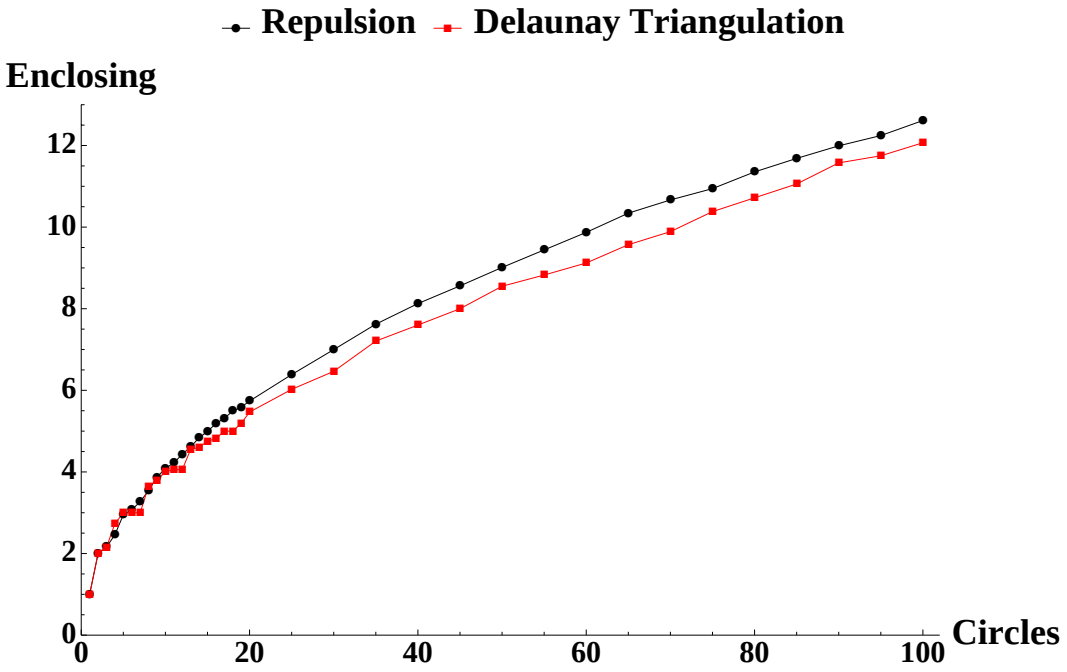


FIGURE 4.7: Comparison between Repulsion-based Repair and Delaunay Triangulation-based Repair by Genetic Algorithms

optimum. This situation occurs when the circles in the optimal configuration form holes involving more than three circles, and sometimes with one or two circles in the hole, i.e. some circles are isolated. This repairing process forces the circles to be tangent, so it is impossible to approach the optimum. In those cases the repulsion-based repair is more feasible to obtain a closer result to the optimum.

In the last test cases, dealing with uneven circles, the expected results were achieved. Starting with no variation, the results depart from the densities reached by the unit-circle problem. As the variance of the circle sizes increases, the density of the solutions increases. The density of the solutions seems to tend asymptotically to a limit which remains to be determined in future experiments.

4.2 Packing Circles into a Rectangular Container

4.2.1 Unit Circle Tests

To measure the performance of the two repair mechanisms I compare the numerical performance of four Metaheuristics (GA, ES, DE, and PSO) applied to the Circle Packing Problem into a Rectangular Container. The most recent solutions to the circle packing

problem are reported at Birgin and Gentil [2010]. This thesis reports proven optimal solutions from 1 to 50 unit circles. To illustrate the effectiveness of our approaches, I tested problems from 1 circles to 50 circles. Only some of them are reported for the sake of brevity. I compare the results produced by our algorithms with the benchmarks presented in Birgin and Gentil [2010].

The metrics used in the comparison of results are area and density. Area is the Best-known solution proven mathematically for a particular problem and density describes the ratio of total area occupied by the circles to container area, i.e., the relationship between the area of the rectangular container and the sum of areas of the circles that are inside it. Benchmarks are registered on table 4.3 in the column labeled *Optimal*, with their corresponding areas and densities.

The results of the two repair mechanisms (Repulsion-based repair and DT-based repair) are presented in sections 4.2.1.1 and 4.2.1.2, respectively. These tables compare our approaches with the benchmarks. The results are shown for the size of the instances, the area of the rectangular container and the density corresponding to the rectangular container of the mean solution obtained with Genetic Algorithms (GA), Differential Evolution (DE), Evolution Strategies (ES) and Particle Swarm Optimization (PSO).

4.2.1.1 Results Using Repulsion-based Repair

Table 4.3 is composed of twelve columns: the first column refers to the number of unit circles to be packed; the second and third columns refer to the area and density, respectively, of the optimal solution; the fourth and fifth columns show the mean area and density obtained applying GA, the sixth and seventh columns show the mean area and density obtained applying DE, the eighth and ninth columns show the mean area and density obtained applying ES, the tenth and eleventh columns show the mean area and density obtained applying PSO, the twelfth column shows the mean time required. The results produced by our approach were obtained by 30 independent executions with random initial populations. The table shows the mean of the 30 executions. The values highlighted in each row represent the best obtained mean.

N	*Optimal*		*GA*		*DE*		*ES*		*PSO*		Time(sec)
	a_0	*density*	a_0	*density*	a_0	*density*	a_0	*density*	a_0	*density*	
1	4.000000	0.785398	**4.000000**	**0.785398**	**4.000000**	**0.785398**	**4.000000**	**0.785398**	**4.000000**	**0.785398**	2
2	8.000000	0.785398	**8.000000**	**0.785398**	**8.000000**	**0.785398**	**8.000000**	**0.785398**	**8.000000**	**0.785398**	34
3	12.000000	0.785398	**12.000000**	**0.785398**	**12.000000**	**0.785398**	12.020136	0.785398	12.001744	0.785398	351
4	16.000000	0.785398	16.053743	0.782769	**16.000208**	**0.785388**	16.037529	0.78356	16.012106	0.784804	754
5	20.000000	0.785398	22.484055	0.698627	**21.919897**	**0.716608**	22.472710	0.698979	22.154969	0.709004	984
6	24.000000	0.785398	24.174964	0.779714	**24.008055**	**0.785135**	24.096489	0.782253	24.048156	0.783825	1580
7	28.000000	0.785398	31.350499	0.701461	**30.572693**	**0.719307**	31.075424	0.70767	31.397415	0.700413	1735
8	32.000000	0.785398	34.226955	0.734297	**32.589747**	**0.771186**	33.719445	0.745349	33.315290	0.754391	2046
9	36.000000	0.785398	37.100399	0.762103	**36.095031**	**0.78333**	36.230197	0.780408	36.287211	0.779182	2428
10	40.000000	0.785398	47.287370	0.664362	**42.130845**	**0.745675**	42.207082	0.744328	43.129056	0.728417	2730
11	43.712813	0.790558	47.976119	0.720307	**47.282318**	**0.730876**	47.279649	0.730917	48.402872	0.713956	2980
12	48.000000	0.785398	47.976119	0.711928	**48.720050**	**0.77379**	50.082998	0.752733	49.289517	0.76485	3430
13	52.000000	0.785398	59.900837	0.681805	**59.464143**	**0.686812**	58.191528	0.701832	60.160156	0.678866	3860
14	54.641016	0.804932	64.413243	0.682815	**60.961710**	**0.721474**	60.398623	0.7282	61.031556	0.720648	4575
15	59.712813	0.789176	71.303353	0.660893	**64.215354**	**0.733841**	61.443445	0.766947	65.589578	0.718466	5280
16	63.444864	0.79227	74.569361	0.674077	**67.812579**	**0.741241**	67.647662	0.743048	72.251408	0.695702	5705
17	65.569219	0.814514	79.935748	0.668125	**78.670002**	**0.678875**	78.899936	0.676896	78.334279	0.681784	6146
18	70.908965	0.797483	84.231600	0.671347	**81.246127**	**0.696017**	81.252912	0.695959	82.158711	0.688286	6755
19	74.641016	0.799698	85.918845	0.694728	**84.737507**	**0.704414**	82.283621	0.725421	83.739387	0.712810	7375
20	76.497423	0.821359	94.386671	0.665686	**93.317953**	**0.673309**	92.043149	0.682635	91.869307	0.683926	7890
25	97.033321	0.804932	119.75300	0.655848	**112.29954**	**0.699378**	114.902588	0.683534	119.024188	0.659864	9765
30	114.746134	0.805188	145.37784	0.648295	**142.038356**	**0.663538**	143.650857	0.656089	143.591565	0.65636	11416
35	131.138439	0.826493	177.46803	0.619581	**165.550574**	**0.664182**	166.624181	0.659903	170.890206	0.643429	13262
40	151.119201	0.811696	199.01109	0.631441	**188.850254**	**0.665415**	192.533964	0.652683	194.634780	0.645638	15124
45	169.387151	0.809411	227.865092	0.620418	**217.276920**	**0.650652**	217.139048	0.651065	217.087798	0.651219	18348
50	185.779455	0.830485	247.545677	0.634548	**236.189970**	**0.665056**	237.438607	0.661559	246.641961	0.636873	22870

TABLE 4.3: Performance using Repulsion based Repair applying GA, DE, ES and PSO

From table 4.3 we can see that DE outperforms the other metaheuristics. This result was at all expected; it is vox populi in the Evolutionary Computation community, that DE is one of the best metaheuristics for real-valued optimization problems. Table 4.3 also shows, in the last column, the computational time required to solve each problem size. Since both GA and DE were run with the same population size and number of generations, their computational times are basically the same.

Figure 4.8 illustrates the plot formed by the mean values obtained for the solution for the four metaheuristics and the benchmarks corresponding for each problem size (i.e. number of circles). Figure 4.9 illustrates the plot formed by the Density obtained by the mean values of the fitness function for the four metaheuristics and the benchmarks corresponding for each problem size (i.e. number of circles). From Figures 4.8 and 4.9 we can see that the best densities were obtained by DE for most cases. Those figures show the distance of ECPP's solutions with respect to the optimal ones.

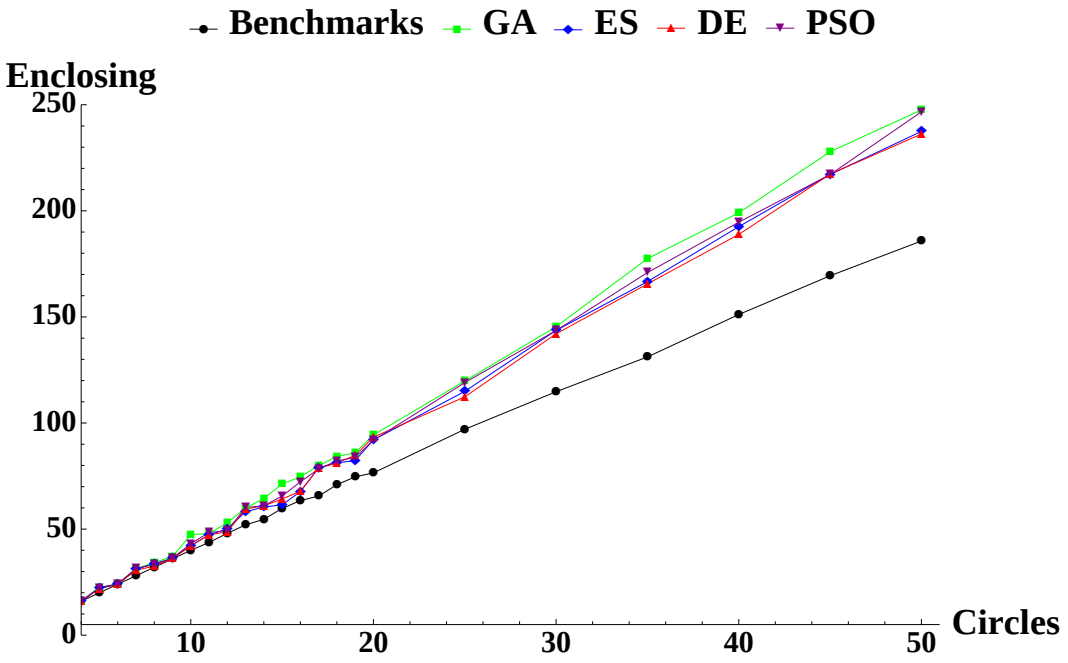


FIGURE 4.8: Area of the Rectangular Container using Repulsion-based Repair

4.2.1.2 Results Using DT-based Repair

Table 4.4 has the same structure as Table 4.3. The results produced by our approach were obtained by 30 independent executions with random initial solutions. The table shows the mean of the 30 executions. The values highlighted in each row represent the

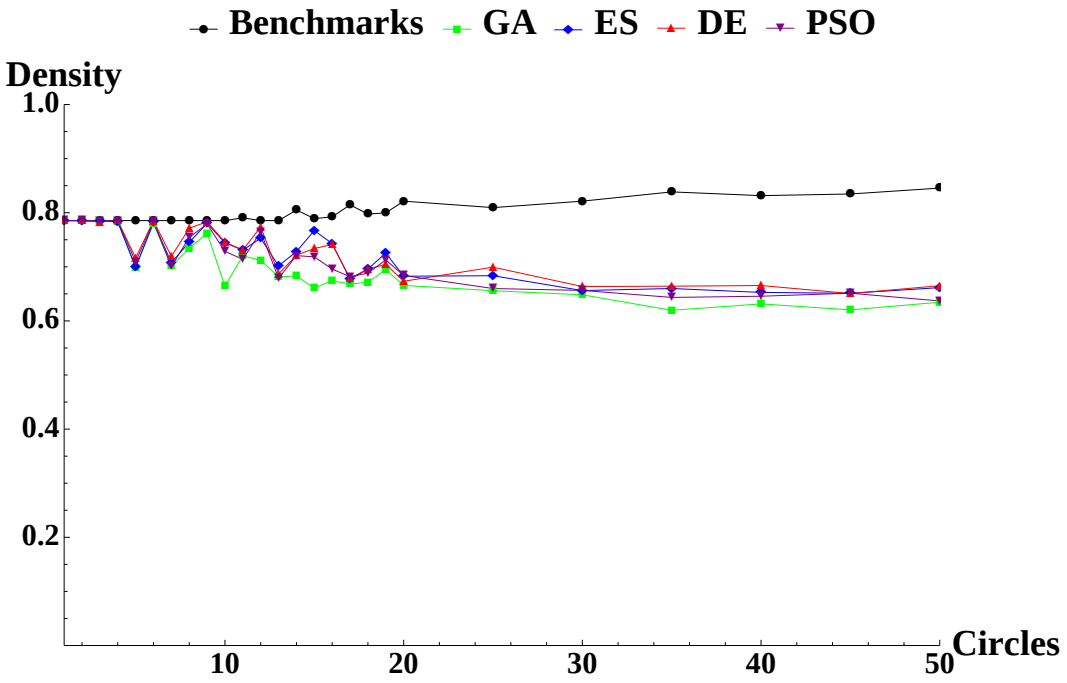


FIGURE 4.9: Density of the Rectangular Container using Repulsion-based Repair

best result obtained.

N	*Optimal*		*GA*		*DE*		*ES*		*PSO*		Time(sec)
	a_0	*density*	a_0	*density*	a_0	*density*	a_0	*density*	a_0	*density*	
1	4.000000	0.785398	**4.000000**	**0.785398**	**4.000000**	**0.785398**	**4.000000**	**0.785398**	**4.000000**	**0.785398**	34
2	8.000000	0.785398	**8.000000**	**0.785398**	**8.000000**	**0.785398**	**8.000000**	**0.785398**	**8.000000**	**0.785398**	34
3	12.000000	0.785398	**14.9282**	**0.631340**	**14.9282**	**0.63134**	**14.92820**	**0.631340**	**14.92820**	**0.631340**	204
4	16.000000	0.785398	**18.6603**	**0.673430**	**18.6603**	**0.67343**	**18.66030**	**0.673430**	**18.66030**	**0.673430**	408
5	20.000000	0.785398	**22.3923**	**0.701489**	**22.3923**	**0.701489**	**22.39230**	**0.701489**	**22.39230**	**0.701489**	696
6	24.000000	0.785398	**26.1244**	**0.721532**	**26.1244**	**0.721532**	**26.12440**	**0.721532**	**26.12440**	**0.721532**	1262
7	28.000000	0.785398	**29.8564**	**0.736564**	**29.8564**	**0.736564**	**29.85640**	**0.736564**	**29.85640**	**0.736564**	1668
8	32.000000	0.785398	**32.7846**	**0.766602**	**32.7846**	**0.766602**	**32.78460**	**0.766602**	**32.78460**	**0.766602**	1970
9	36.000000	0.785398	**38.2487**	**0.739223**	**38.2487**	**0.739223**	**38.24870**	**0.739223**	**38.24870**	**0.739223**	2284
10	40.000000	0.785398	**41.0526**	**0.765261**	**43.1769**	**0.727609**	**41.05260**	**0.765261**	**43.17690**	**0.727609**	2524
11	43.712813	0.790558	**43.7128**	**0.790558**	**43.7128**	**0.790558**	**43.71280**	**0.790558**	**43.71280**	**0.790558**	2892
12	48.000000	0.785398	**49.1769**	**0.766602**	**49.1769**	**0.766602**	**49.17690**	**0.766602**	**49.17690**	**0.766602**	3164
13	52.000000	0.785398	**54.641**	**0.747437**	**54.641**	**0.747437**	**54.64100**	**0.747437**	**54.64100**	**0.747437**	3542
14	54.641016	0.804932	**54.641**	**0.804932**	**57.5692**	**0.76399**	**54.64100**	**0.804932**	**57.56920**	**0.763990**	4175
15	59.712813	0.789176	**60.1051**	**0.784025**	**64.7654**	**0.727609**	**60.10510**	**0.784025**	**64.76540**	**0.727609**	4511
16	63.444864	0.79227	**64.7654**	**0.776117**	**64.7654**	**0.776117**	**64.76540**	**0.776117**	**64.76540**	**0.776117**	4955
17	65.569219	0.814514	**71.4256**	**0.747730**	**71.4256**	**0.74773**	**71.42560**	**0.747730**	**71.42560**	**0.747730**	5124
18	70.908965	0.797483	**71.9615**	**0.785818**	**76.4974**	**0.739223**	**71.96150**	**0.785818**	**76.49740**	**0.739223**	5448
19	74.641016	0.799698	**79.1577**	**0.754068**	**79.1577**	**0.754068**	**79.15770**	**0.754068**	**79.15770**	**0.754068**	5829
20	76.497423	0.821359	**79.1577**	**0.793756**	**79.1577**	**0.793756**	**79.15770**	**0.793756**	**79.15770**	**0.793756**	6014
25	97.033321	0.804932	**98.2102**	**0.799711**	**98.2102**	**0.799711**	**98.21020**	**0.799711**	**98.21020**	**0.799711**	8490
30	114.746134	0.805188	**123.923**	**0.760535**	**123.923**	**0.760535**	127.92300	0.7367540	127.92300	0.7367540	9616
35	131.138439	0.826493	**148.708**	**0.739409**	**142.851**	**0.769722**	148.70800	0.7394090	149.24400	0.7367540	10276
40	151.119201	0.811696	**159.904**	**0.785871**	**159.904**	**0.785871**	169.49200	0.7414130	173.49200	0.7243190	11867
45	169.387151	0.809411	**185.885**	**0.760535**	**185.885**	**0.760535**	197.74100	0.7149340	197.74100	0.7149340	12885
50	185.779455	0.830485	211.865	0.741413	**210.669**	**0.745622**	211.86500	0.7414130	210.66900	0.7456220	15777

TABLE 4.4: Performance using Delaunay Triangulation based Repair applying GA, DE, ES and PSO

From table 4.4 we can see that DE outperforms the other metaheuristics. This result was at all expected; it is vox populi in the Evolutionary Computation community, that DE is one of the best metaheuristics for real-valued optimization problems.

Figure 4.10 illustrates the plot formed by the mean values obtained for the solution for the four metaheuristics and the benchmarks corresponding for each problem size (i.e. number of circles). From Figure 4.10 we can see that the best solutions were obtained by DE.

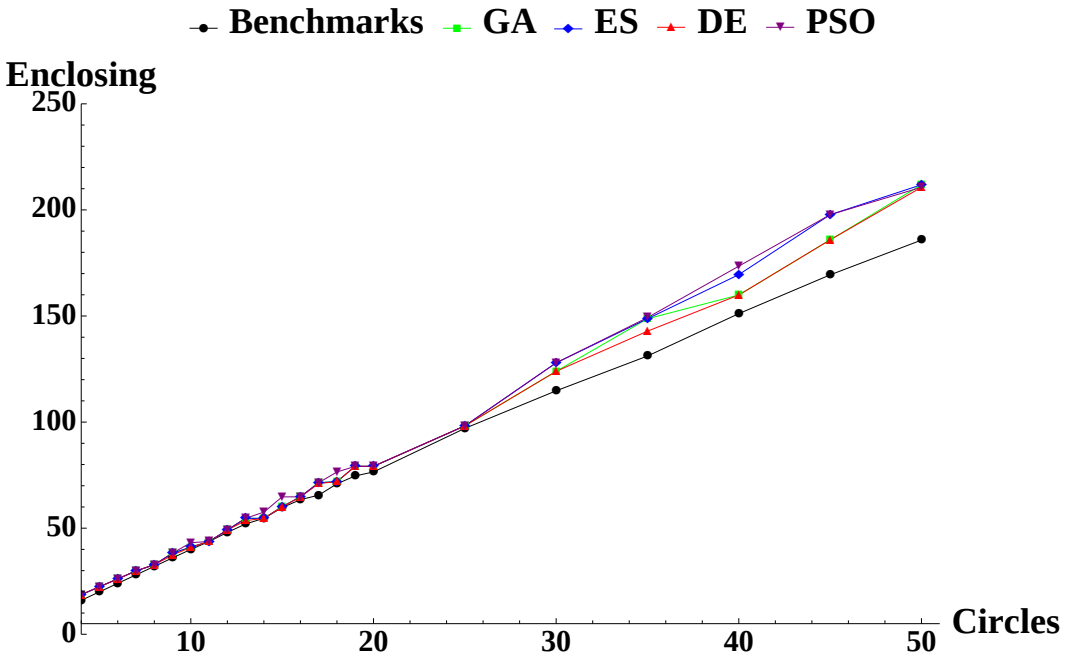


FIGURE 4.10: Area of the Rectangular Container using Delaunay Triangulation-based Repair

Figure 4.11 illustrates the plot formed by the Density obtained by the mean values of the fitness function for the four metaheuristics and the benchmarks corresponding for each problem size (i.e. number of circles). From Figure 4.11 we can see that the best densities were obtained by DE.

4.2.2 Discussion

Comparing the four meta-heuristics using Delaunay Triangulation-based Repair, we can see that Differential Evolution perform best, followed closely by Genetic Algorithms,

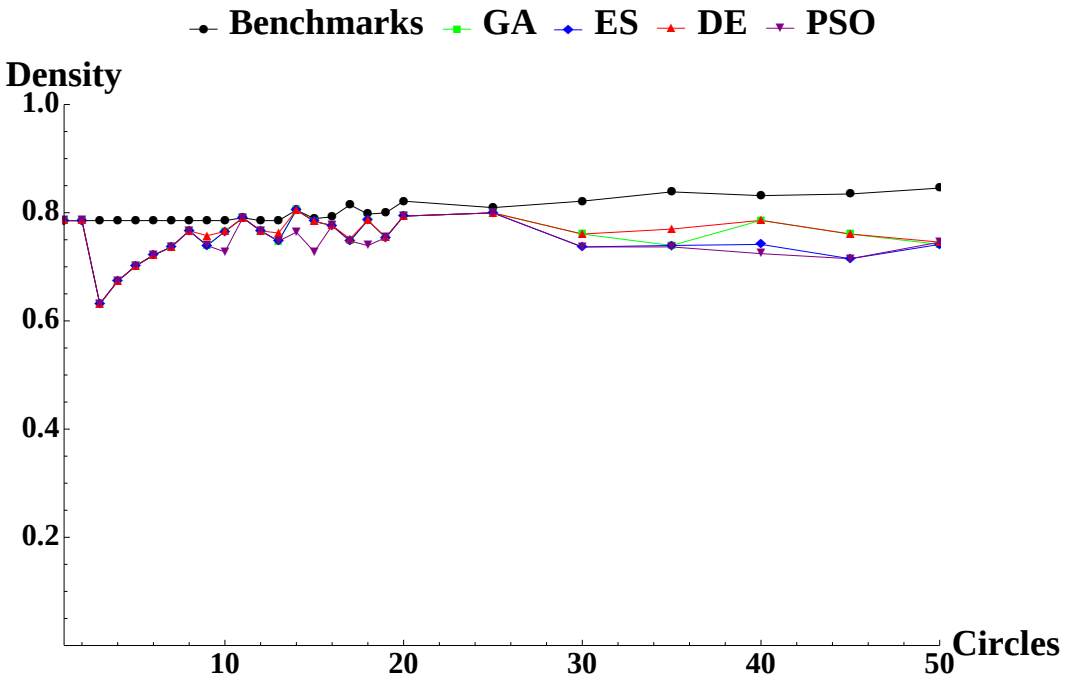


FIGURE 4.11: Density of the Rectangular Container using Delaunay Triangulation-based Repair

Evolutionary Strategies being further back, and in last place Particle Swarm Optimization.

As we can see in Figures 4.10 and 4.11 the difference between Differential Evolution and Genetic Algorithms is minimal and perhaps one might think that these results do not show obvious differences.

To compare the results obtained by the Repulsion and Delaunay Triangulation based repair processes I plot the mean of size of the area of the rectangular container for the different problem sizes. Figure 4.7 shows that the Delaunay Triangulation repair performs better than the Repulsion-based repair. Also, the repairing process by Delaunay Triangulation requires less time; that means less evaluations are needed to obtain a result closer to the optimum, than by the repulsion heuristic.

In very few cases, where the optimal configuration is a configuration that can not be obtained by the Delaunay Triangulation-based repair, this repair method has trouble approaching the optimum. This situation occurs when the circles in the optimal configuration form holes involving more than three circles, and sometimes with one or two circles in the hole, i.e. some circles are isolated. This repairing process forces the circles

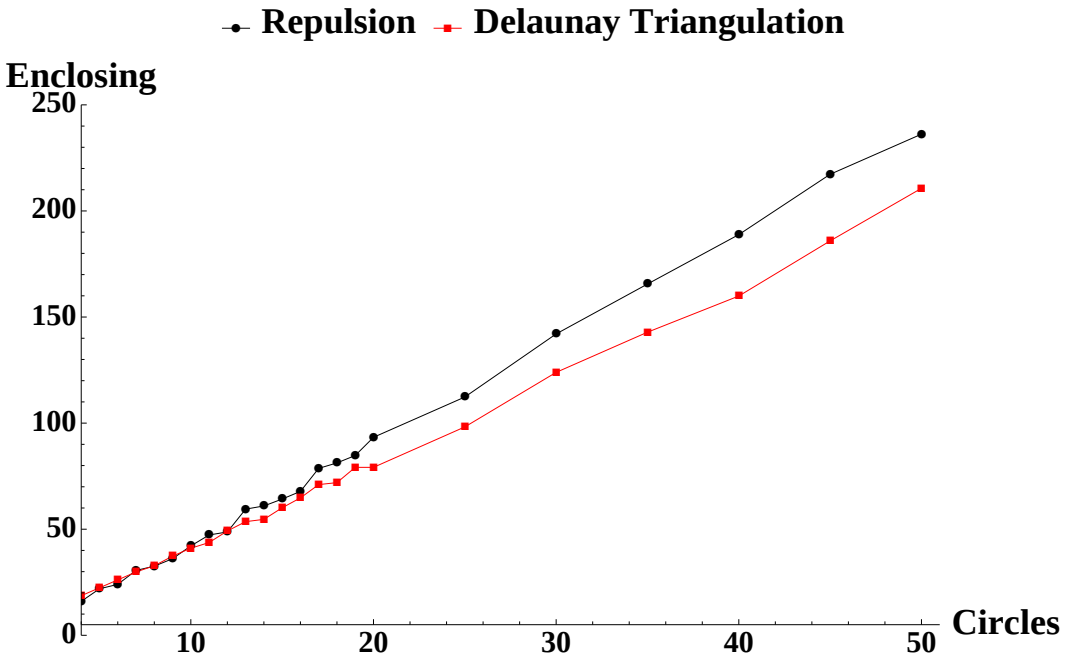


FIGURE 4.12: Comparison between Repulsion-based Repair and Delaunay Triangulation-based Repair by DE

to be tangent, so it is impossible to approach the optimum. In those cases the repulsion-based repair is more feasible to obtain a closer result to the optimum.

4.2.3 Final Remarks

This chapter presents the results of four metaheuristics used for the enclosing container that could be circular or rectangular. To obtain this container firstly a repair mechanism must be applied to ensure that the given configuration of the set of circles does not violate any restriction. To repair a solution, two algorithms can be used, the repulsion-based repair algorithm and the Delaunay triangulation-based repair algorithm.

Both repair algorithms produce feasible configurations, but as we can see in Tables 4.1 to 4.4 the time consumed by Delaunay triangulation-based repair is smaller than that of Repulsion-based repair. Furthermore, the results are better using Delaunay triangulation-based repair as Figures 4.7 and 4.12 show, exept on those cases where the optimal solution is not a configuration where all circles are tangent to one another in triplets, i.e. a set of 4, 5, or 6 unitary circles, or where exist at least one circle that must be isolated, i.e. a set of 8 or 9 unitary circles. This exception explains the oscillations in Figure 4.11, where the solution found for one problem size is good and for the next size it is worse.

Chapter 5

Conclusions

This thesis proposes an Evolutionary Computation-based solution to the Circle Packing Problem, called ECPP. These solutions are based on two repair mechanisms: repulsion-based and Delaunay Triangulation-based. I combine these repair heuristics with GA, ES, DE, and PSO. The solution space of the circle packing problem is enormous and increases rapidly with the problem size, i.e., the number of circles to be allocated. Since these packing problems are NP-complete, heuristic search procedures are used.

The strength of evolutionary algorithms lies in the ability to search large and complex solution spaces in a systematic and efficient way. Evolutionary search strategies are not dependent on a particular problem structure and allow the user to use different methods for the encoding of the genotype. The performance of the search process is strongly related to the representation of the circle packing problem. The particular feature of our evolutionary algorithm developed for the circle packing problem is their two stage approach. ECPP is used to explore and manipulate the solution space, and a second procedure is used to evaluate the solutions. The genotype needs to be repaired in order to check the quality and feasibility of the packing solution: the phenotype.

The operation on the layout rather than an encoded data structure raises a number of other issues, such as overlap. Overlapping configurations are invalid solutions and need to be resolved either by rejecting, correcting, or temporarily accepting them. Rejection wastes precious computation time and may result in less dense layouts for highly uneven radii, since the slightest change in position or rotation could lead to invalid configurations, which will no longer contribute to the search process; after the repair process is applied to a prospect solution, it results in a valid solution. Correcting invalid configurations seems a better option, since often only minor repositioning is necessary to obtain

a valid solution.

The acceptance of an invalid layout requires a penalty term in the evaluation function, as mentioned in the solutions presented in the literature. The penalty expression needs to be carefully designed balancing between layout compaction and overlap generation. Penalty functions are a less efficient guide to the search than a repair algorithm that avoids producing constraint violating configurations.

When the search process operates on an encoding, the packing rules applied by the decoding algorithm guarantee that all solutions considered in the search process are valid. There has been much speculation on whether this is beneficial with respect to the transmission of specific layout to the next generation and the next state in the neighborhood, respectively. We have not been able to find a satisfactory answer to this problem in the literature. The different solution approaches have not been compared with each other. Since much of their performance strongly depends on radii and the number of circles involved with respect to the formulation of the objective function, it is not sufficient to judge their performance purely on the basis of benchmarks achieved. This emphasizes the need for including density in the relative evaluation of the solutions presented in this work.

The numerical results show that all methods can find acceptable solutions and their performances will be very close if the Delaunay Triangulation repair is used. We can also observe that Differential Evolution performs better than the other three metaheuristics when a repulsion-based repair is applied and Genetic Algorithms performs better when a Delaunay Triangulation-based repair is applied.

Our findings clearly show that ES and PSO provide the worst results of the four metaheuristics with the two repair mechanisms, on most of the problem instances. GA and DE got better results, using either of the repair mechanisms. In some cases the repulsion-based repair gives us better results than the DT-based repair. This occurs particularly because the configuration of the best known solution is not formed by triplets of circles tangent to each other. Nonetheless, in general terms, the performance of DT-based repair is better, specifically given that this repair process forces the circles to be tangent with at least two other circles. Probably a hybrid algorithm composed with the two repair mechanism could give us better results than those obtained with each heuristic separately, specifically if I apply the repulsion-based repair followed by the DT-based

repair.

Finally, the last set of experiments show that ECPP is able to handle problem instances with circles of different sizes, as effectively as for the unit circle instances. This result was expected, since the solution was designed without any size-related constraint. ECPP was compared with the results published at MathRec. Those results represent the state of the art in solving that specific problem instance, and no single algorithm provides the best solution to all problem sizes. Even more, not all participants provided solutions to all problem instances. In general, algorithms are best at a single solution or at most at a range of them. Furthermore, none of the solutions presented in the MathRec contest was based on Evolutionary Computation; to the authors' knowledge, ECPP is the first attempt to solve CPP using metaheuristics of these kind.

Future work will focus on the packing of polygons on rectangles and strips. So far, We have not found any Evolutionary Computation based solution to the CPP that had been previously published, therefore, We cannot objectively compare with other results of metaheuristic search methods applied to solve CPP.

Bibliography

Addis, B., Locatelli, M., and Schoen, F. (2008). Efficiently packing unequal disks in a circle. *Operations Research Letters*, 36(1):37–42.

Akeb, H., Hifi, M., and M'Hallah, R. (2009). An adaptive beam search look-ahead algorithm for the circular packing problem. *Revised for International Transactions in Operational Research*, 17:553–575.

Akeb, H., Hifi, M., and Negre, S. (2011). An augmented beam search-based algorithm for the circular open dimension problem. *Computers and Industrial Engineering*, 61:373–381.

Al-Modahka, I., Hifi, M., and M'Hallah, R. (2011). Packing circles in the smallest circle: an adaptive hybrid algorithm. *Revised for Journal of the Operational Research Society*.

Albano, A. and Sapuppo, G. (1980). Optimal allocation of two-dimensional irregular shapes using heuristic search methods. *Systems, Man and Cybernetics, IEEE Transactions*, 10:242–248.

Birgin, E. G. and Gentil, J. M. (2010). New and improved results for packing identical unitary radius with triangles, rectangles and strips. *Computer and Operational Research*, pages 1318–1327.

Carrabs, F., Cerrone, C., and Cerulli, R. (2014). A tabu search approach for the circle packing problem. *17th International Conference on Network-Based Information Systems*.

Castillo, I., Kampas, F. J., and Pintér, J. D. (2008). Solving circle packing problems by global optimization: numerical results and industrial applications. *European Journal of Operational Research*, 191(3):786–802.

Deb, K. (2004). A population-based algorithm-generator for real-parameter optimization. *Soft Comput*, 9:236–253.

Demaine, E. D., Fekete, S. P., and Lang, R. J. (2010). Circle packing for origami design is hard. *CoRR*, 2.

Dowsland, K. A., Gilbert, M., and Kendall, G. (2007). A local search approach to a circle cutting problem arising in the motor cycle industry. *Journal of the Operational Research Society*, 58(4):429–438.

George, J. A., George, J. M., and Lamar, B. W. (1995). Packing different-sized circles into a rectangular container. *European Journal of Operational Research*, pages 693–712.

Graham, R. L. and Lubachevsky, B. D. (1996). Repeated patterns of dense packings of equal disks in a square. *The Electronic Journal of Combinatoric*, 3.

Haims, M. J. and Freeman, H. (1970). A multistage solution of template-layout problem. *Systems Science and Cybernetics, IEEE Transactions*, 6:145–151.

Harary, F., Randolph, W., and Mezey, P. G. (1996). A study of maximum unit-circle caterpillars–tools for the study of the shape of adsorption patterns. *Discrete Applied Mathematics*, 67(1-3):127–135.

Hifi, M. and M'Hallah, R. (2003). Approximate algorithms for constrained circular cutting problems. *Computers and Operations Research*, 31(5):675–694.

Hifi, M. and M'Hallah, R. (2009a). Beam search and non-linear programming tools for the circular packing problem. *International Journal of Mathematics in Operational Research*, 1(4).

Hifi, M. and M'Hallah, R. (2009b). A literature review on circle and sphere packing problems: Models and methodologies. *Advances in Operations Research*, 2009(150624).

Hifi, M., Paschos, V. T., and Zissimopoulos, V. (2004). A simulated annealing approach for the circular cutting problem. *European Journal of Operational Research*, 159:430–448.

Huang, W. and Chen, M. (2006). Note on: an improved algorithm for the packing of unequal circles within a larger containing circle. *Computers and Industrial Engineering*, 50(3):338–344.

Lee, D. and Schachter, B. J. (1980). Two algorithms for constructing a delaunay triangulation. *International Journal of Computer and Information Sciences*.

Lu, Z. and Huang, W. (2008). Perm for solving circle packing problem. *Computers & Operations Research*, 35(5):1742–1755.

Machado, P. and Leitao, A. (2011). Evolving fitness functions for mating selection. *Lecture Notes in Computer Science*, 6621.

Michalewicz, Z. (1996). Genetic algorithms+datastructures=evolutionary programs. *Springer-Verlag, third edition*.

Mladenovic, N., Plastria, F., and Urosevic, D. (2005). Reformulation descent applied to circle packing problems. *Computers & Operations Research*, 32(9):2419–2434.

Nordbakke, M. W., Ryum, N., and Hunderi, O. (2004). Curvilinear polygons, finite circle packings, and normal grain growth. *Materials Science and Engineering A*, 385(1-2):229–234.

Pintér, J. D. and Kampas, F. J. (2006). Nonlinear optimization in mathematica with math-optimizer professional. *Mathematica in Education and Research*, 10:1–18.

Specht, E. (1999). Packomania web site. http://www.packomania.com/.

Stoyan, Y. G. and Yas'kov, G. (1998). Mathematical model and solution method of optimization problem of placement of rectangles and circles taking into account special constraints. *International Transactions in Operational Research*, 5:45–57.

Stoyan, Y. G. and Yas'kov, G. (2004). A mathematical model and a solution method for the problem of placing various-sized circles into a strip. *European Journal of Operational Research*, 156:590–600.

Sugihara, K., Sawai, M., Sano, H., Kim, D. S., and Kim, D. (2004). Disk packing for the estimation of the size of a wire bundle. *Japan Journal of Industrial and Applied Mathematics*, 21(3):259–278.

Szabó, P. G., Markót, M. C., and Csendes, T. (2005). Global optimization in geometry - circle packing into the square. *Essays and Surveys in Global Optimization*, pages 233–265.

Szabó, P. G., Markót, M. C., Csendes, T., Specht, E., Casado, L. G., and García, I. (2007). New approaches to circle packing in a square: With program codes. *Springer Optimization and Its Applications*, 6.

Wang, H., Huang, W., Zhang, Q., and Xu, D. (2002). An improved algorithm for the packing of unequal circles within a larger containing circle. *European Journal of Operational Research*, 141(2):440–453.

Wenqi, H. and Yan, K. (2004). A short note on a simple search heuristic for the diskspacking problem. *Annals of Operations Research*, (1-4):101–108.

Xu, Y.-C., XIao, R.-B., and Amos, M. (2007). A novel genetic algorithm for the layout optimization problem. *Evolutionary Computation 2007 CEC 2007*, pages 3938 – 3943.

Yan-Jun, S., Yi-Shou, W., Long, W., and Hong-Fei, T. (2012). A layout pattern based particle swarm optimization for constrained packing problems. *Information Technology Journal*, 11:1722–1729.

Yan-jun, S., Zhuang-Cheng, L., and Shuai, M. (2010). An improved evolution strategy for constrained circle packing problem. 6215:86–93.

Zhang, D. F. and Deng, A. S. (2005). An effective hybrid algorithm for the problem of packing circles into a larger containing circle. *Computers & Operations Research*, 32(8):1941–1951.

Zhi-Qin, Q., Hong-Fei, T., and Zhi-Guo, S. (2001). Human-computer interactive genetic algorithm and its application to constrained layout optimization. *Chinese Journal of Computers*, 5:553–560.

Zimmermann, A. (2006). Al zimmermann's programming contests. http://www.recmath.org/contest/CirclePacking/index.php.