Masters' Degree in Informatics Engineering
Dissertation
Final Report

Swarm Intelligence Algorithms for Cluster Geometry Optimization

Nuno António Marques Lourenço
naml@student.dei.uc.pt

Advisors:
Francisco B. Pereira
Luís Paquete

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

This work was supported by Fundação para a Ciência (FCT), Portugal, under project PTDC/QUI/ 69422/2006, which is financed by Programa Operacional Factores de Competitividade (COMPETE) of QREN and FEDER programs. The author is grateful to the Laboratory for Advanced Computing of the University of Coimbra for the provision of supercomputer time on Milipeia (Project daion_10).

Acknowledgements

I am grateful to my advisors, Francisco B. Pereira and Luís Paquete, who have supported me with their knowledge throughout this dissertation while allowing me the room to work in my own way.

I would like to express my regards to the ECOS (Evolutionary and Complex Systems) group for the contributions and opinions.

I would like to show my gratitude to my colleagues, particularly Diogo Machado, Fábio Pedrosa, Ivo Gonçalves and Marco Simões for all the time that we have spent together, and the constant encouragement.

To my family and friends.

To all people who contributed to my education.

My final words are addressed to my parents. Thank you for the support and encouragement. Without them I would not be who I am today and this work would not be possible.

Nuno Lourenço
Coimbra, July 2011

Abstract

The problem of cluster geometry optimization is relevant for many areas from protein structure prediction to the field of nanotechnology. A cluster is an aggregate of interacting atoms or molecules and it can hold a few or even millions of elements. Finding the organization for the atoms/molecules that has the lowest potential energy is an NP-hard problem. In this dissertation we propose an approach based on Swarm Intelligence algorithms. In particular we describe the application of an algorithm based on Ant Colony Optimization to the cluster geometry optimization problem. Results are promising, since the the proposed approach is able to discover almost all the best-known solutions for short-ranged Morse clusters between 30 and 50 atoms. A comparative analysis with some state-of-art algorithms is presented and it shows that our approach can be as effective as the state-of-art algorithms. Moreover we perform an experimental analysis to understand the effect of algorithms components in the overall performance.

Keywords: Cluster geometry optimization, Morse Cluster, Swarm Intelligence, Ant Colony Optimization

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Challenging optimization problems occur in many science and engineering fields. In the Chemistry field, the problem of finding the lowest energy configuration of an atomic/molecular cluster is one example of these type of problems. For example, it seems that the native structure of a protein is related with the lowest energy configuration of atoms that composes it. If the structure could be effectively and reliably derived from the amino-acid sequence, this knowledge would provide new insights into the nature of protein folding. Such insights would be helpful, for example, for the development of drugs by pharmaceutical companies [12].

The problem of finding the lowest energy configuration of an atomic/molecular cluster is usually designated by *Cluster Geometry Optimization*. It addresses how should a set of atoms/molecules be placed in a 3-dimensional space, to get the minimal configuration energy. During the last years, some efforts have been made to develop effective algorithms for this type of problem.

In this dissertation we present Discrete Ant Colony Cluster Optimization (DACCO), a discrete Ant Colony Optimization algorithm, for the problem of cluster geometry optimization. Ant Colony algorithms are powerful metaheuristics for discrete optimization and, in this work, we propose a novel approach to apply them to continuous domains. Firstly, DACCO discretizes the domain. Then, the ants belonging to the colony build possible solutions in the discrete domain. Thirdly, these solutions are moved back to the continuous space using a local optimization procedure and are evaluated. This process is repeated for several iterations.

To complement this work we review the state-of-art approaches to the problem, by focusing in algorithms that have been applied to the cluster geometry

optimization problem.

1.1 Contributions

The contributions of this work can be summed up into into three main points:

- We propose an ant colony algorithm that discretizes a continuous problem and allows the application of well-known discrete variants to solve it. The approach comprises a set of mechanism that help to map solutions from one space into another.
- Second, the proposed approach was implemented and tested. The resulting framework can be parameterized, and it has diagnostic tools in order to assess what is happening during the evolution process.
- Finally, we performed a set of experiments with the proposed approach. These experiments allowed us to make some conclusions about its effectiveness. The conclusions focus on the individual results, and on the comparison with other approaches proposed in the literature.

1.2 Structure of the dissertation

The remainder of this document is as follows: In the next chapter we introduce the problem of cluster geometry optimization. In Chapter 3 we describe some optimization algorithms, and highlight the main achievements in what concerns their application to cluster geometry optimization. In Chapter 4 we detail the DACCO framework. Chapter 5 presents results of the application of DACCO to the problem of cluster geometry optimization. Finally, Chapter 6 gathers the main conclusions and points towards possible further work.

Chapter 2

Cluster Geometry Optimization Problem

In this chapter we describe the problem of cluster geometry optimization. In Section 2.1 we describe atomic clusters, and some of their properties. In Section 2.2 we introduce the potential used to measure the interactions of the elements in the cluster.

2.1 Atomic Clusters

Understanding the properties of chemical clusters is relevant in many scientific fields, from protein structure prediction to the field of the nanotechnology [12]. In simple terms a cluster is a set of a few to millions of atoms or molecules, which may present distinct properties from those of a single particle. In order to describe the interactions that occur in the cluster a multidimensional function is used. This function, known as the Potential Energy Surface (PES), contains all the relevant information about the chemical system, and models all the interactions between the aggregate particles [39].

The goal of cluster geometry optimization is to determine the optimal structural organization for a set of particles that compose an aggregate. In other words, the main goal is to discover the position of all the atoms, or molecules, in a 3D space so that it corresponds to the lowest potential energy.

Since the PES are computationally heavy functions, model PES are adopted when studying large clusters. Examples of simplified PES are the pairwise

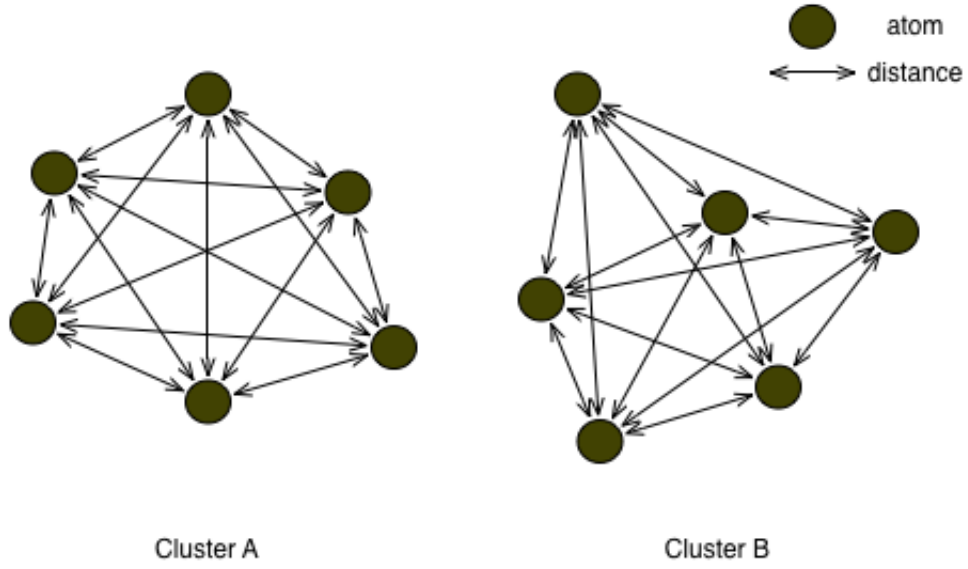

Figure 2.1: Calculation of the pairwise potential for two clusters with 6 atoms: Distances considered to calculate the potential energy for each conformations are specified.

additive potentials that only consider the distance between every pair of particles composing the cluster to determine the energy of the cluster. For a given number of particles different, conformations on the 3-Dimensional space usually lead to different energy values. An example, for a cluster with 6 atoms, is shown in Fig. 2.1.

Usually PES functions define highly roughed landscapes, with multiple valleys [35]. It has been proved that global minimization of atomic PES is a NP-hard problem [14, 38]. Moreover the number of local minima increases exponentially as the cluster grows in size.

2.2 Morse Clusters

As described above, model PES are regularly adopted to understand chemical properties of real materials and as benchmarks of new optimization algorithms. The most widely adopted pairwise models are the Lennard-Jones [21] and Morse potentials [28]. Since the Morse potential provides accurate

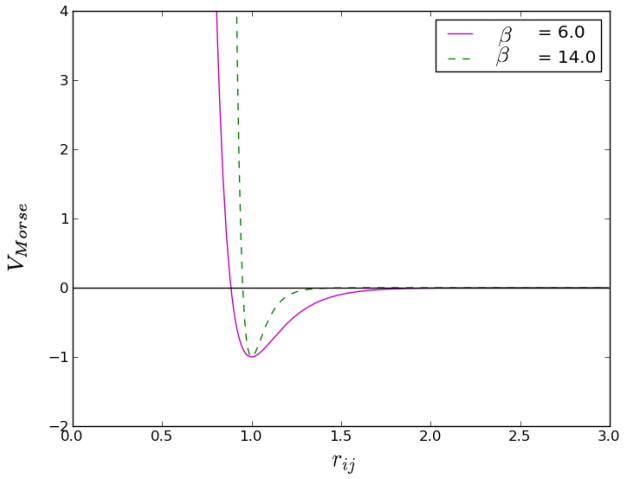

Figure 2.2: Morse Potential for different values of β

approximations of real materials, and define a more challenging benchmark [3, 32], we focus our attention on the later. The energy function of a N-atom Morse cluster is obtained by the sum of all pairwise contributions that occur between the atoms. Following [13, 28] we can formulate this as:

$$V_{Morse} = \epsilon \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left(\exp^{-2\beta(r_{ij}-r_0)} - 2\exp^{-\beta(r_{ij}-r_0)} \right) \tag{2.1}$$

where r_{ij} is the distance between particles i and j in the aggregate, ϵ is the bond dissociation energy, r_0 is the equilibrium bond and β is the range exponent of the potential. Following [13], ϵ and r_0 are both set to 1.0, leading to a scaled version of the Morse function without specific atom interactions. Thus, the potential has a single parameter β that establishes the shape of the energy contribution of every pair of atoms [11]. Fig. 2.2 illustrates how the pairwise contribution is modeled as a function of distance between atoms. Two different values of β are depicted: $\beta = 6.0$, which corresponds to the long-ranged potential, and $\beta = 14.0$, which corresponds to a short-ranged version. As we can see, in both cases, the optimal potential energy is achieved when the atoms are placed at a distance of 1. This value corresponds to the equilibrium bond value. Fig.2.2 also reveals that if we move from a long-ranged potential ($\beta = 6.0$) to a short-ranged version ($\beta = 14.0$), we get a narrower potential curvature, promoting the appearance of roughed search landscapes, with a higher number of local minima [11].

Chapter 3

Optimization Algorithms

In this chapter we describe some approximation methods for global optimization. We do not aim to provide a comprehensive presentation of optimization algorithms. We offer just a brief description of several methods that focus on global, unbiased, iterative, approximation algorithms (a class usually known as *metaheuristics*). The reason for this choice is that current state-of-art methods for cluster geometry optimization belong to this class. In Section 3.1 we briefly highlight single-solution methods, followed by Evolutionary Algorithms in Section 3.2. In Section 3.3, we describe example of swarm intelligence algorithms, focused on ant colony optimization methods. Finally, in Section 3.4, we give a brief description of how some of the presented methods were applied to the problem of cluster geometry optimization.

3.1 Single-solution Methods

Single-solution algorithms generate one initial solution and try to improve it iteratively. A remarkable example is Simulated Annealing in which has been applied to the cluster geometry optimization problem. However, it is not the only single-solution method. Hill-Climbing (HC), Tabu Search (TS), Iterated Local Search (ILS) or Variable Neighborhood Search (VNS) are also other examples of single-solution methods. For details on these methods, the reader can see, e.g., [25].

Simulated Annealing

Simulated Annealing (SA) [19] is a generic probabilistic metaheuristic. The inspiration to this method comes from annealing in metallurgy, a technique involving heating and controlled cooling of a material. The heat causes the atoms constituting the material to become unstuck from their initial positions (a local minimum) and wander randomly through space. Then, the slow cooling gives them more chances of gradually finding configurations with a lower internal energy than the initial one.

Analogously with this physical process, each step of the SA selects a nearby solution. If the new solution is better than the current one, the new solution replaces the current. However, if the new solution is worse than the current it can still be selected. Acceptance depends of two criteria:

1. How worse is the new solution.
2. How many steps have been performed.

The number of steps of the algorithm is controlled by T, which is called temperature. It starts with an initial high value, that decreases over time, until it reaches the minimum value of T_{min}. The generic behavior is shown in Algorithm 1.

Algorithm 1 Simulated Annealing

```
define a random initial solution s
best_solution ← s
while termination condition not met do
  choose a neighbor solution snew
  if accept(s,snew,T) then
    s ← snew
  end if
  if is_better(best_solution,s) then
    best_solution ← s
  end if
  decrease(T)
end while
return best_solution
```

The *accept()* procedure checks if the new solution is to be accepted. *is_better()* check if the current solution is better than the best solution found until that moment.

3.2 Evolutionary Algorithms

Evolutionary Algorithms (EA) are a family of algorithms whose main inspiration is the biological evolution [15]. EAs use the reproduction, mutation, recombination and selection mechanisms, which play an important role in biological evolution. EAs maintain a population of individual solutions, which can be selected for reproduction. Selection is probabilistic and biased by the fitness of the individuals. In reproduction, two selected individuals exchange information about their solutions. This process is called crossover and it creates two new solutions. Since the parents are promising solutions, with crossover we aim to create two new solutions that, hopefully, are promising as well. Next, mutation slightly modifies the offspring. The application of the mutation operator is probabilistic and it aims to maintain diversity in the population. In the last step, offsprings replace the old population, and a new iteration starts. The general algorithm is presented in Algorithm 2.

Algorithm 2 General Evolutionary Algorithm

```
randomly generate initial population
evaluate initial population
while termination condition not met do
  select parents
  crossover
  mutation
  evaluate offspring
  replacement
end while
return best individual in the population
```

For more details on evolutionary algorithms see, e.g., [15].

3.3 Swarm Intelligence

The idea of swarm intelligence is to develop algorithms that model the behavior of a group of animals that engage in social interactions. Some examples are synchronous bird flocking, ants gathering food or fish shoals escaping from a predator [18]. In the following sections we describe the Ant Colony Optimization (ACO) framework, which is inspired by the behavior of foraging ants searching for food.

Traveling Salesman Problem

The Traveling Salesman Problem (TSP) will be used to help explain the ACO algorithms described. We choose this problem, due to its relevance, and because it is a benchmark for all ACO algorithms.

The TSP is the problem of a salesman, who starting from a initial city, wants to find the shortest tour that visits a set of cities. More formally, we have a fully connected, undirected graph $G = (C, E)$, where C contains the cities that the salesman has to visit, and E are the arcs connecting the cities. The problem consists of finding the minimum length Hamiltonian circuit of the graph, where an Hamiltonian circuit is a closed path visiting each node $c \in C$ exactly once.

Ant Colony Optimization Algorithms

Ant colonies are distributed systems [10] composed by a large amount of simple agents. They maintain a highly structured social organization, which allows the colonies to accomplish complex tasks. The individuals in the colony communicate with each other, via *stigmergy*, which is a form of indirect communication mediated by modifications in the environment where the ants are working. In concrete, they have a chemical called pheromone that they deposit in the environment, which will be sensed by other ants, and help them in their work. One example of pheromones is the *"trail pheromone"*, that is very important for the social life of some ant species. This specific type of pheromone allows ants to mark paths, for example, from the nest to food source.

The idea behind ant algorithms is then to use a form of *"artificial stigmergy"*, to coordinates artificial ant colonies. With this in mind, Marco Dorigo proposed an ACO algorithm in 1992 [7].

Algorithm 3 presents the general idea behind the ACO algorithm.

Algorithm 3 General ACO Algorithm

```
while termination condition not met do
  ConstructSolutions()
  UpdatePheromones()
  DaemonActions() {optional}
end while
```

In ACO a problem is represented as a graph G that artificial ants will cross

when building solutions. G is composed composed by a set of solution components C, connected by a set E of edges. The specific composition of G depends on the problem being solved. Using the TSP as an example, the cities that the salesman has to visit correspond to C and the routes that he has to travel correspond to E. In the *ConstructSolutions()* procedure, each ant navigates the graph G, and constructs a solution. This solution is built incrementally by stochastic local decisions that make use of pheromone trails and heuristic information about the problem. Once an ant has built a solution, or while the solution is being built, it evaluates the (partial) solution.

The *UpdatePheromones()* is the process by which the pheromone trails are updated. The trails values can either increase, as ants deposits pheromone in the components or edges that they use, or decrease, due to pheromone evaporation. In practice, the deposit of pheromone trails increases the probability that those components/edges, which belonged to good solutions, will be selected again. Differently, pheromone evaporation implements a forgetting method: it avoids early convergence to a suboptimal region, therefore favoring exploration of new areas of the search space.

Finally *DaemonActions()* procedure corresponds to actions that are not performed by ants. Examples of this daemon actions are the local search procedures, or the collection of global information to be used whether it is useful or not to deposit additional pheromone to bias the search process from a non local perspective.

During the years several ACO algorithms were proposed. Next we introduce and describe the some of these proposed algorithms.

Ant System

Ant System (AS) was first proposed in 1992 by Marco Dorigo [7, 9]. Initially, the pheromone trails are initialized according to a simple rule: the initial value must be higher than the total amount of pheromone that an ant will deposit in one iteration. Heuristics to calculate this value are commonly used.

Each ant in AS constructs a complete solution to the problem in question. For instance, in the TSP, an ant must construct a tour. At each construction step, an ant uses a probabilistic rule to decide which new component it should add to the current solution. Following [9], the probability with which ant k, currently at city i, chooses to go to city j is given as follows:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\gamma}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\gamma}, \quad j \in N_i^k, \tag{3.1}$$

where τ_{ij} is the pheromone trail in edge (i, j), η is an heuristic value that is problem related (e.g. for the TSP: $\eta = \frac{1}{d_{ij}}$, where d_{ij} is the distance between city i and city j), N_i^k is the feasible neighborhood when the ant k is in the node i (e.g. in TSP is the set of cities that ant k has not visited yet), α, γ are parameters of the algorithm. The α parameter determines the relative influence of the pheromone trail, and γ determines the relative influence of the heuristic information.

After all ants have constructed their solutions, the pheromone trails must be updated. First, and if we consider pheromone evaporation [9], we must update the trail by taking into account the percentage of pheromone to evaporate:

$$\tau_{ij} = (1 - \rho)\tau_{ij}, \quad \forall (i, j) \in E, \tag{3.2}$$

where $0 \leq \rho \leq 1$ is a parameter that determines the quantity of pheromones to evaporate.

Then we perform the pheromone trails update [9]:

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k, \quad \forall (i, j) \in E, \tag{3.3}$$

where m is the total number of ants, $\Delta\tau^k$ is calculated using $\frac{1}{L^k}$, where L^k is the fitness of the solution of the k^{th}-ant.

This algorithm had a poor performance when applied to larger instances of optimization problems [10], like the TSP. Therefore, there have been attempts to improve the algorithm. Elitist Ant System(EAS) [7, 9], Rank-Based Ant System(AS_{rank}) [4], Ant Colony System [8], and Max-Min Ant System (MMAS) [36] are the most relevant contributions.

Elitist Ant System

The main idea behind Elitist Ant System (EAS) [7, 9] is that the edges belonging to the best solution found since the beginning of the algorithm should receive more pheromones. Adding an additional parameter that reinforces pheromones if an edge belongs to the best tour so far to Eq. (3.3) [7, 9], the final equation will be:

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k + e\Delta\tau^{bs}, \tag{3.4}$$

where e is the weight given to the best tour so far, and $\Delta\tau^{bs}$ is determined using $\frac{1}{L^{bs}}$, where L^bs is the fitness of the best solution so far.

Rank-Based Ant System

In AS_{rank} each ant deposits pheromones according to its rank, when compared to other ants of the colony. Besides the best-so-far-ant deposits an additional amount of pheromone.

Before the updating of pheromones, ants are sorted, by ascending order, of the quality of the current solution. In each iteration, only the $(w-1)$ best ranked ants, and the best ant so far, can deposit pheromone, where w is a parameter. Thus, the AS_{rank} pheromone update rule is as follow [4]:

$$\tau_{ij} = \tau_{ij} + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs}, \tag{3.5}$$

.

Ant Colony System

This algorithm [8], despite of being inspired by the AS, differs from it in three main aspects:

1. It exploits the accumulated search experience, through the use of a more aggressive action choice rule: Eq. 3.6.
2. Pheromone update and pheromone deposit takes place only on the edges belonging to the *best-so-far* solution;
3. It uses an online pheromone update strategy. Each time an ant, while building solutions, uses an edge (i, j), it removes some pheromone from it. This increase the exploration of alternative solutions.

In Ant Colony System(ACS), when an ant is building a solution, it chooses to move from component i to component j according to the following *pseudorandom proportional* rule [8]:

$$j = \begin{cases} \arg\max_{l \in N_i^k}(\tau_{il}[\eta_{il}^{\gamma}]), & \text{if } q \leq q_0 \\ P, & \text{otherwise,} \end{cases} \tag{3.6}$$

where q is a uniformly distributed variable in the range $[0, 1]$, $q_0, 0 \leq q_0 \leq 1$, is a parameter, P is a random variable selected according to Eq. (3.1), and γ is a parameter. The q_0 parameter defines the degree of exploration and the choice of whether to concentrate the search around the *best-so-far*, or explore other locations.

$\mathcal{MAX} - \mathcal{MIN}$ Ant System

$\mathcal{MAX} - \mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) introduces four main modifications to the traditional AS [36]:

1. only the *best-so-far* or the *iteration-best* ants are allowed to deposit pheromones:
$$\tau_{ij} = \tau_{ij} + \Delta\tau_{ij}^{best} \tag{3.7}$$
where $\Delta\tau^{best}$ is the computed using $\frac{1}{L^{best}}$, L^{best} is the fitness of the best solution so far or the fitness of the best solution in the iteration, depending of which one is used. In most cases *best-so-far* and *iteration-best* according to a schedule, that can be based, for example, in the number of iterations.

2. Introduction of a limit to pheromone trails $[\tau_{min}, \tau_{max}]$. This modification intends to counteract the effect of stagnation, that is the situation where all ants build the same solution.

3. In the beginning, all the pheromone trails are initialized with τ_{max}. Thus, and with small pheromone evaporation rate, the exploration is increased in the early stages of the search.

4. Pheromone trails are reinitialized every time the system approaches stagnation, or when no improved tours have been generated for a certain number of consecutive generations.

Multi-colony ant algorithm

The main idea behind this algorithm is to have a set of identical colonies solving the same problem. In [27] they used a variant of ACS with multiple colonies working in different locations of the search space, where the best colonies share information with the worst. This information sharing is made by passing along, to the worst colony, the *best-so-far* solution. Every time a migration occurs, the worst colony uses the new solution as if it was found by it.

Approximate Nondeterministic Tree Search

Approximate nondeterministic tree search (ANTS) is an ACO algorithm which exploits the ideas from mathematical programming [26]. ANTS computes lower bounds every time it adds a new component to a partial solution, and then uses it to define the heuristic information necessary to build solutions. The use of this lower bounds has advantages, since we can discard a component if it leads to partial solutions that have a larger cost than the best-so-far solution. However calculating lower bounds at each step, can introduce a higher computational overhead.

Hyper-Cube Framework for Ant Colony Optimization

The hyper-cube framework was introduced in [2] to automatically rescale the values of pheromone trails, so they can stay in range $[0, 1]$. This choice was inspired by other algorithms that used binary representation of the solutions. In this type of representation, the decision variables typically correspond to solution components as they are used by the ants, that is, if a component is in the ant solution, it takes the value 1, else it takes the value 0. Thus, a solution corresponds to one corner of a *n-dimensional* cube, where n is the number of decision variables. The relationship with ACO lies in the normalization of the pheromones to the interval [0,1]. In this case, the pheromone vector is a point in the solution space; in case τ is a binary vector, which corresponds to a solution of the problem.

Ant Colony Optimization for continuous domains

ACO algorithms were originally proposed to solve discrete problems. Yet there are problems that are not discrete. Thus, and trying to keep the biological inspiration, there are some proposals to solve continuous problems using an ACO framework. Here we present one of the main approaches, proposed by Socha et al.[33]. For other alternatives in ACO for continuous problems see [1, 20, 37].

Ant Colony Optimization for continuous and Mixed-Variable Optimization

ACO for continuous and Mixed-Variable Optimization($ACO_{\mathbb{R}}$) uses a simple approach: if a discrete probability distribution is used for discrete problems,

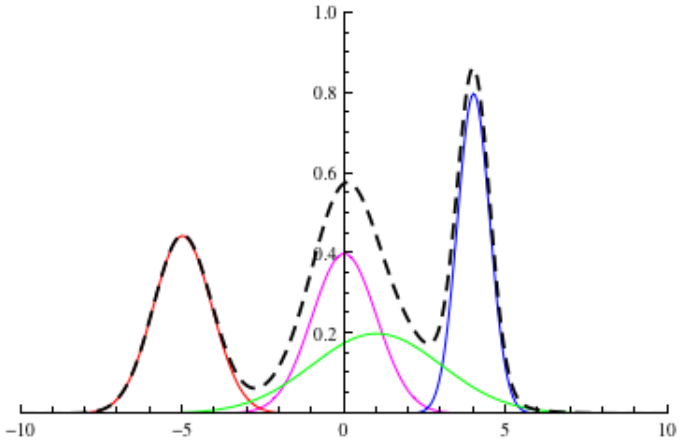

Figure 3.1: Example of a mixture of gaussian kernels - The dashed line is mixture of all the other depicted gaussian functions

then a continuous probability distribution function - the *Probability Density Function* (PDF), should be adopted for continuous problems [33].

The most popular PDF is the normal (or gaussian) function, defined by,

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{3.8}$$

where μ is the mean and the σ is the variance. The use of the normal distribution has one drawback, because it can not describe a situation where there are two promising disjoint locations in the search space, as it only has one maximum. To solve this problem, the algorithm uses a mixture of normal distributions, as we can se in Fig. 3.1.

While building solutions, an ant, at a certain step i, generates a random number according to an PDF that composes the kernel mixture, and add it to the solution. This process is repeated until a complete solution has been constructed. Then solutions are evaluated.

After the evaluation process we have to update the pheromone values. Hence we have to reinforce the influence of functions that lead to good solutions, and reduce the the influence of the function that lead to not so good solutions. The reinforcement can be achieved by adding a new PDF function, and the decrease can be achieved by removing PDFs from the kernel mixture. For more details in the algorithm see [33, 34].

3.4 Cluster Geometry Optimization

Here we present some algorithms that were proposed in the literature to tackle the problem of Cluster Geometry Optimization, using the Morse potential as a benchmark function.

One of the first attempts of global optimization of the Morse potential was made by Wales et al. [12, 13]. In their work they used a Monte Carlo minimization method [22] combined with a local search algorithm. This method is known as Basin-Hopping (BH). In the work conducted by Wales et al., they deform the surface of the PES, in a way that some of the valleys are removed. This transformation does not affect the energies of the minima. They were able to found almost all the known global optima for short-ranged ($\beta = 14.0$) Morse clusters up to 80 atoms.

Roberts et al. [30] proposed the first EA to the optimization of Morse clusters. They used a certain number of components that had been previously proposed: real-valued representation of the Cartesian coordinates of the atoms, a local search method to improve the solutions, and the Cut and Splice [6] crossover. This crossover is a specific operator handling atomic clusters. First, it defines a plane that passes through the center of mass of the clusters to be combined. Then the clusters are cut by this plane and the complementary halves are joined (spliced) together in order to form a new offspring. The mutation operator in this work corresponds to assign random coordinates to a certain number of atoms. The algorithm was applied to instances between 19 and 50 of the Morse Potential with $\beta = 6.0$ and $\beta = 14.0$. The results showed that it was able to discover nearly all the known global optima [30].

Grosso et al. [16] proposed Population Basin Hopping (PBH), a stochastic method, in which different conformations of the clusters are maintained in a population of solutions, hence ensuring a sufficient level of diversity. The diversity is enforced through the definition of problem specific diversity measures. Moreover, in their work, Grosso et el. applied a two-phase local search algorithm, in order to improve the efficiency of the method.

Dynamic lattice searching (DLS) is another effective approach for cluster geometry optimization [5]. It starts with a randomly generated local minimum and iteratively applies a greedy strategy to search for better solutions in the neighborhood. This search is aid by a Dynamic Lattice (DL). The DL is constructed adaptively, based on a starting local minimum, and then locating all the possible locations for new atom. Afterwards, they perform a greedy search for conformations with low potential energy values: it iteratively moves the

atom located at the position with the highest energy (in the lattice) to the vacant position with the lowest energy (in the lattice) [5, 31]. The method is restarted several times to ensure proper exploration of the search space [5].

Pereira et al.[29] proposed a steady-state EA, with diversity control to the problem of cluster geometry optimization. In a steady-state EA, after generating a new offspring it is necessary to decide if it will enter the population, and if it does, which individual it will replace. In general, replacement strategies are related with the fitness and/or the age of individuals. Fitness based strategies check the quality of the individuals to decide who will be replaced. In the age based strategies the oldest elements are replaced. Moreover, these strategies can still be combined with other mechanisms, in order to keep a suitable level of diversity in the populations. In Pereira et al.[29] they propose a couple of mechanisms to keep the diversity in the population. They showed that the diversity is important to improve the effectiveness of the EA. The proposed EA was able to find all the known optima for short-ranged Morse clusters up to 80 atoms.

More recently, Lourenço et al. [24] proposed a Particle Swarm Optimization algorithm to the problem of cluster geometry optimization. The proposed algorithm has several features that were specifically design to tackle this problem. The most noteworthy are: the adoption of specific rules to update current position of particles, where the velocity is only applied to fraction of the solution; and the embracing of a steady-state strategy to update the population particles. This strategy allows for a simultaneous exploration of the neighborhoods of the current locations, and of the best ever seen solutions. Additionally it has a diversity mechanism, to postpone convergence. The results presented showed that the algorithm was able to find all the best-known solutions to short-ranged Morse instances between 30 and 50 atoms.

Chapter 4

DACCO: Discrete Ant Colony Cluster Optimization

In this chapter we present DACCO, the ACO algorithm that we used to tackle the problem of Cluster Geometry Optimization. In this dissertation we are interested in developing an effective ACO approach to the problem of cluster geometry optimization. Yet, ACO algorithms were officially proposed for discrete environments and currently there are many variants that are state-of-art methods for different combinatorial optimization problems. Despite a few research efforts [1, 20, 33, 37], existing ACO algorithms to continuous domains are somehow incipient, particularly if compared with the most well-known discrete variants such as $\mathcal{MAX} - \mathcal{MIN}$, Ant System or Ant Colony System. Hence, in our research we decided to discretize our problem in order to use a discrete variant of ACO. Fig. 4.1 presents an overview of the framework that we propose.

A mapping operator converts the original search space into a discrete version. Then, the ACO algorithm builds the solutions in the *Discrete-Space*. After, the application of *Local Optimization* converts the solutions back to the *Continuous Space* where they are evaluated. Finally, the *Mapping* operator will convert the evaluated solutions to the discrete space to allow the pheromone update.

This framework poses some interesting research questions that will be addressed in the next sections:

1. How to model a real-valued problem in a such a way that it can be solved by a discrete ACO algorithm (*Mapping*);

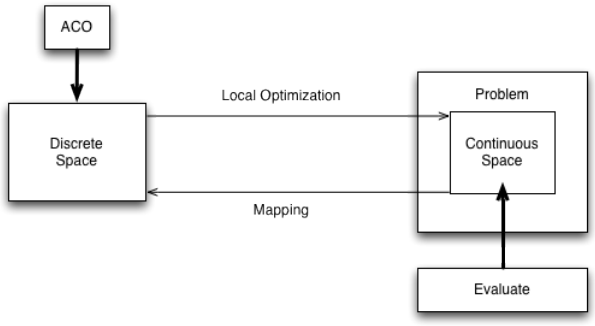

Figure 4.1: Overview of DACCO framework

2. Study the performance of an ACO algorithm in a problem that was not originally discrete;

DACCO follows the $\mathcal{MAX} - \mathcal{MIN}$ Ant System [36], since it was one of the most successful ACO approaches. However, $\mathcal{MAX} - \mathcal{MIN}$ has some drawbacks:

1. It might be difficult to define the initial values to the pheromone limits;
2. Readjust the limits every time a new best solution is found;
3. The decision in what ant to use to update the pheromones;

To overcome these drawbacks, DACCO incorporates the modifications proposed by the Hyper-Cube Framework (HCF) [2]. In the HCF the pheromones values are always kept in the interval [0,1]. Hence, we do not have to define an initial limit to the pheromones neither readjust them every time we find a new best solution. Furthermore, the rule to update pheromones in [2] uses more than one ant. They use weights to adjust the relative influence of each ant to the pheromone values. These weights depend on the state of the algorithm.

DACCO is not an exact $\mathcal{MAX}-\mathcal{MIN}$ / HCF variant as it incorporates some modifications, in order to adapt it to our problem. The main modifications are:

1. The pheromone update rule does not deposit all the pheromone in one position;
2. We only use two ants to update the pheromones.

In the next sections we present a high level overview of our entire algorithm. Thereafter we will split it in its main components and we will explain them in

more detail.

4.1 DACCO

Here we present a high level description of our ACO algorithm for cluster geometry optimization: *DACCO*. In the following sections, we break it into smaller pieces, and explain them in detail.

Algorithm 4 DACCO

```
Construct Search Space
Initialize Pheromones
while termination condition not met do
  Construct Solutions
  Evaluate Solutions in Continuous Space
  Convert Solutions to Discrete Space
  Apply Discrete Local Search
  Update Pheromone Values
end while
return best individual in the population
```

4.1.1 Construction of Search Space

In this procedure we discretize the search space, so that we can build a graph G where the artificial ants can work. The search space is defined by a cube of size $N^{(1/3)}$, where N is the number of atoms, as depicted in Fig. 4.2. To transform it we decided to divide the cube of Fig. 4.2 into smaller cubes to which we gave the name of *cells*. These cells have to be large enough to contain one atom, but small enough to avoid having more than one atom. The final result of this transformation is depicted in Fig. 4.3.

The Alg. 5 gives more details about the construction of the space. It receives the cube size $N^{(1/3)}$, and the cell size W. In the first instruction we begin by defining an empty search space. Then we define the maximum coordinate of our problem. It is important to say that our cube is only defined in the non-negative side of the xx, yy, zz axis (Fig. 4.2). After this we calculate the center of each cell in terms of (x,y,z) components and add it to the new search space. The return value is the discretized search space.

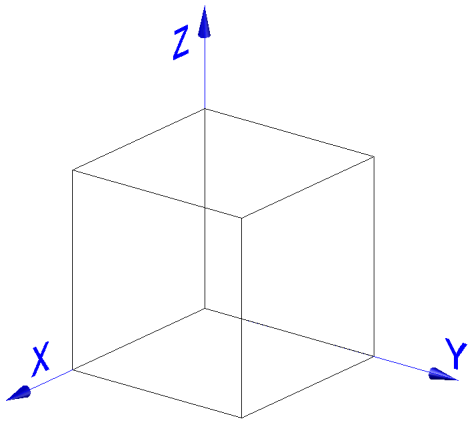

Figure 4.2: Search Space.

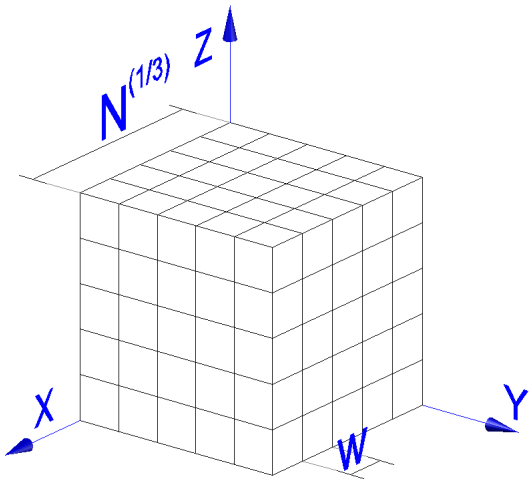

Figure 4.3: Division of the search space in cells.

The cells that are part of the discretized search space are then used as components C of G that the ants use to build solutions. The edges E are arcs connecting all the components of the graph. In the following sections we will use the name *cells* indistinctively.

4.1.2 Initialize Pheromones

Before we start the optimization process, we need to initialize the pheromone matrix. This matrix contains the information about the quantity of pheromones of a certain path, and therefore it plays an important role in process of building solutions. For example, in the TSP the pheromone matrix contains information about how many times a certain path from city i to city j has appeared in a good solution.
In cluster geometry optimization problems what is important is the position of atoms in space, and not the order in which they are placed. Therefore in DACCO, we are not interested in building paths, but place atoms in cells. Thus, instead of having information about how many times we have gone from cell i to cell j, we have information about the desirability of having a certain component i in a solution (i.e. placing an atom in a cell i).

In the beginning of the algorithm, all the cells have the same desirability of being in a solution, and for that reason we initialize the pheromone matrix with the value of 0.5, as in [2].

Algorithm 5 Construct Search Space

```
discretized_search_space = {}
max_coord = N^(1/3)
cell_center = W/2
total_number_of_cells = ceil(max_coord/W)
for x = 1 → total_number_of_cells do
  for y = 1 → total_number_of_cells do
    for z = 1 → total_number_of_cells do
      discretized_search_space+ = (x * W + cell_center, y * W +
      cell_center, z * W + cell_center)
    end for
  end for
end for
return discretized_search_space
```

4.1.3 Construction of Solutions

The construction of solutions is achieved by the method Construct Solutions. An ant solution corresponds to a complete atomic cluster. The algorithm starts by defining a start position for the ant, thus we have to place the ant in a cell of our search space. Then the ant calculates which are the neighbors of it, and chooses, in a probabilistic way, a new cell to move. After it knows which cell is the next, it places the atom in the center of the current cell, adds it to solution, and moves to next one. This process is repeated until each ant has a cluster with all the atoms in place. The general algorithm that defines this construction process is detailed in Alg. 6.

Algorithm 6 Construct Solutions

```
Given an ant do:
placed_elements = 1
while placed_elements < N do
  neighbors = find_feasible_neighborhood(ant)
  next_cell = find_next_cell(ant, neighbors, pheromone_matrix)
  ant.solution[placed_elements] = Atom(ant.current_cell)
  ant.current_cell = next_cell
  placed_elements = placed_elements + 1
end while
```

In the end of Alg. 6 an ant will have built a complete atomic cluster.

In the following we explain:
the $find\_feasible\_neighborhood()$ and $find\_next\_cell()$.

Find Feasible Neighborhood

This procedure determines the cells that are available in the neighbor of the current position of an ant. To find these neighbors, we used some different techniques:

1. Moore Neighborhood
2. Convex Hull
3. Full Moore Neighborhood

Moore Neighborhood This neighborhood is defined, in $\mathbb{R}^3$, by the cube that is centered in a the current cell (x0, y0, z0). The Moore neighborhood of range r can be defined by the set M of all points (x, y, z) that verify the following condition:

$$M = \{(x, y) : |x - x0| \leq r \wedge |y - y0| \leq r \wedge |z - z0| \leq r\} \tag{4.1}$$

where $r \geq 0$. However, in our approach the values of r are only in the range $r > 0$, once we need to have at least one cell that is different from (x0, y0, z0). Fig. 4.4 depicts the a Moore neighborhood with $r = 1$. We use a 2D representation for simplicity.

The final M set will then be used to determine which cell will be the next.

Convex Hull With this technique we wanted to use more information about the problem, in the choice of the neighbors. Since the Morse potential takes in account the distance of all pair of atoms that composes the aggregate, we thought that we should use the information of the partially constructed solution to help choose the neighborhood. One of the first ideas that came into mind was the following: build a convex hull, with the atoms that are in the partial solution, and then determine all the cells that are at distance one from the segments defined by the points of the convex hull. The process is detailed in Alg. 7.

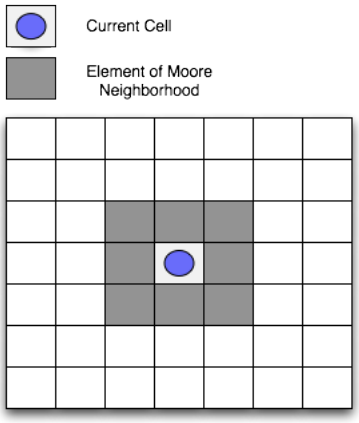

Figure 4.4: The grey cells represent the final M set, with $r = 1$

Algorithm 7 Convex Hull

```
Given a partial solution of an ant do:
ch = find_convex_hull(partial_solution)
for i = 1 → length(ch) do
  for all cells CL of discretized_search_space do
    if (distance(CL, segment(ch[i − i], ch[i])) = 1 then
      neighbors = neighbors ∪ CL
    end if
  end for
end for
M = remove_repeated_cells(neighbors)
return M
```

The *remove_repeated_cells*() procedure removes the repeated cells that are in the M set, since one cell can be in more the one atom neighborhood. The M set will then be used to determine which cell should be added to the solution.

This technique uses the information of all the atoms that had already been placed, but it demands a lot of computational resources. Every time we place a new atom we have to calculate the Convex Hull of the current partial solution. This, together with the additional time on the calculus of the distance of all cells to the segments, we decided to consider other alternatives.

Full Moore Neighborhood This technique was another attempt to use information about the partial constructed cluster. Here we iterate by all atoms

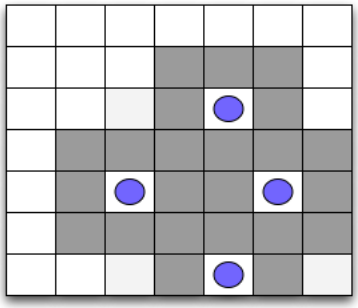

Figure 4.5: The grey cells represent the final M set, with $r = 1$

that are in the partial solution, and we look for the neighbors of them. The neighborhood that we use is the Moore neighborhood with the same r for all the atoms. The general idea is depicted in Fig. 4.5. We use a 2D representation for simplicity. The following algorithm gives more details about this technique:

Algorithm 8 Full Moore Neighborhood

```
Given a partial solution of an ant do:
for i = 1 → length(ch) do
  for all atoms AT of partial solution do
    neighbors = neighbors ∪ Moore_Neighborhood(AT)
  end for
end for
M = remove_repeated_cells(neighbors)
return M
```

Find Next Cell

This procedure receives the feasible neighborhood and determines the next cell to be part of the solution. This means that, an ant, located in a cell i should choose a new cell j to add to the solution. This choice is based on the pheromone value of cell j, τ_j^α and the heuristic information η_j^γ. It is important to say that we do not use the heuristic component of the choice function. The heuristic information is applied when we are building the feasible neighborhood. In the case of our problem we want to place atoms that are close to each other, and being so we can discard the cells that are too far away from the current one. Taking this into account the choice function depends

only of the pheromone value of cell j:

$$p_j = \frac{[\tau_j]^\alpha}{\sum_{l \in M} [\tau_l]^\alpha} \tag{4.2}$$

Providing all values of the rule function are stored in an certain set V, the iterative cell selection is determined by a roulette wheel algorithm.: each value of the V determines a slice on a circular roulette wheel. Next, the wheel is spun and the cell to which the marker points is chosen as the next cell for the ant. In Alg. 9 we detail the find next cell procedure, and in Alg. 10 we detail the roulette wheel selection method.

Algorithm 9 Find Next Cell

Given a set M of candidate cells **do:**
for all cells in M **do**
  $V =$ **Apply Eq. 4.2 the probability of each cell**
end for
return $roullete\_wheel(V)$

Algorithm 10 Roulette wheel

Given a set V of probabilities **do:**
$index = 0$
$total = V[0]$
pick a random value r **uniformly from** $[0, 1]$
while $total < r$ **do**
  $index = index + 1$
  $total = total + V[index]$
end while
return $index$

After finding the next cell, the ant will place the atom in its center, and then repeat all this process until it has a complete solution, that is, all the atoms have been placed.

4.1.4 Evaluate Solution in Continuous Space

The evaluation of the solutions built by the ants is made in this procedure. Since we have the atoms in the center of the cell, which are represented by (x, y, z), where x, y, z $\in \mathbb{R}$, we do not have to make any additional computation to pass from the discretized space to the continuous space.

The evaluation of the solutions proceeds in two steps. First, the Broyden-Fletcher-Goldfarb-Shannon (L-BFGS) quasi-Newton method [23] move the solution to the nearest local optimum. Then equation 2.1 is used to determine the potential energy of the resulting cluster.

L-BFGS is an efficient local optimization method that combines the modest storage and computational requirements of conjugate gradient methods with the super linear convergence exhibited by full memory quasi-Newton strategies. This local optimization algorithm is usually adopted by hybrid approaches for cluster geometry optimization problems [16, 17, 29].

4.1.5 Convert Solutions to Discrete Space

In this procedure we convert the solutions, which were returned by Evaluate Solutions in Continuous Space procedure, to the discrete space. This is very important, because the solutions returned after their evaluation are different from the ones that were given as parameters. To convert the cluster to the discretized space, we apply the following algorithm: given a certain atom, we move it to center of the nearest cell. This process is detailed in Alg. 11.

Algorithm 11 Convert Solution to Discrete Space

Given a solution CS in the continuous space **do:**
for all atoms AT **of** CS **do**
  for all cels CL **of** *discretized space* **do**
    $distances[AT][CL] = distance(AT, CL)$
  end for
  $discrete\_solution = discrete\_solution \cup min\_dist\_cell(AT, distances)$
end for
return *discrete_solution*

The *min_dist_cell*() procedure receives an array distance of atom to cells, and returns the cells that is closer to the atom in question.

After this, we move all the cluster to the origin of the axis. This allows us to have the information concentrated in a place of the search space.

4.1.6 Apply Discrete Local Search

This procedure is responsible for applying some perturbations in the solutions that were built by the ants, in an attempt to improve the search results.

After the solutions have been converted to the discrete space, we look for the atom that has the worst contribution to the Morse potential, and we move it to a random cell. After this, we evaluate the solution again, and if the result is a better solution, we keep it. This process is repeated for a given number of tries. The Alg. 12 gives a description of the process:

Algorithm 12 Apply Discrete Local Search

```
Given an ant solution ASdo:
i = 0
best_solution = AS
current_solution = AS
while i < local_search_iterations do
  worst_atom_cell = find_atom_worst_contribution(current_solution)
  new_position = random_cell()
  current_solution[worst_atom_cell] = new_position
  evaluate(current_solution)
  if is_better(current_solution, best_solution) then
    best_solution = current_solution
  else
    current_solution = best_solution
  end if
end while
return best_solution
```

The *local_search_iterations* are the number of iterations that we make to try improve the solutions. The *random_cell*() procedure returns a random cell that belongs to the search space, and that is not already occupied.mThe *is_better*() procedure checks if the first solution is better than the second one.

4.1.7 Update Pheromone Values

This procedure is responsible for the update of the pheromones. We start by decreasing the current values of the pheromones, in a certain percentage, simulating the pheromone evaporation in the nature. After we follow the rule presented in [2] to deposit pheromones, with some minor differences. In the cited approach, they used 3 ants to update the trails:

1. *Iteration-best ant* - which corresponds to the best ant in the current iteration of the algorithm;

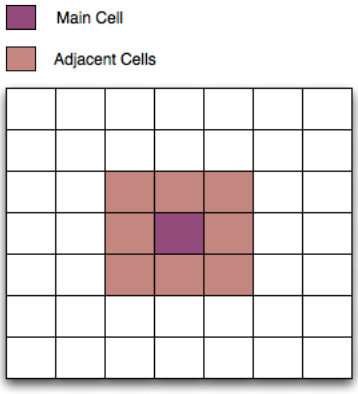

Figure 4.6: Pheromone Propagation

2. *Restart-best ant* - which corresponds to the best ant found since the restart of the algorithm
3. *Global-best ant* - which corresponds to the best solution found since the start of the algorithm

However, in our approach we do not have a restart mechanism, thus we do not need to use the restart-best ant.

As was pointed out by [2], finding a schedule for the usage of the ants can be a difficult task, and it requires a lot of experimentation. However in the HyperCube Framework, we do not have this problem, because both iteration-best and global-best ants are allowed to deposit pheromones. The influence of each ant is measured by the weights that we assign to each one. The final rule to update pheromones is depicted as follows:

$$\tau = \tau + \omega \tag{4.3}$$

where

$$\omega = w_{ib} * F_{ib} + w_{gb} * F_{gb} \tag{4.4}$$

where w_{ib} is the weight of the iteration-best ant, F_{ib} is the quality of the iteration-best ant, w_{gb} is the weight of the global-best ant, F_{gb} is the quality of the global best ant, and $w_{ib} + w_{gb} = 1$.

After the update is applied, pheromone values that exceed τ_{max} are set back to τ_{max}. A similar process is applied to τ_{min}.

The final important aspect of the pheromone update rule is that the pheromones are not deposited in only in one cell. Instead, we only deposit a percentage p in the main cell, and we propagate a percentage $(1 - p)$ to the adjacent cells (Fig. 4.6). This propagation is uniformly distributed among all the adjacent cells. This means that the value that is given to each one of the adjacent cells is $(1 - p)/(number\ of\ adjacent\ cells)$. Such aspect is important to smooth the transitions between components in the search space.

In appendix A we provide some technical decisions related with the implementation of this approach.

Chapter 5

Results

In this chapter we present an experimental analysis of the application of our algorithm to several instances of short-ranged Morse Clusters. To perform such analysis, we start by describing the scenario used in our experiments. In Section 5.2 we present and discuss the results obtained by the DACCO algorithm. In Section 5.3 we present a comparative analysis between DACCO and other approaches of the literature. Finally, in Section 5.4 we present a detailed analysis of some main components of the algorithm.

5.1 Experimental Scenario

In this study we focus our attention in several instances of short-ranged Morse clusters. More precisely, we selected instances with a number of atoms that ranges between 30 and 50. With this scenario we aim for two goals:

1. Assess the performance of the algorithm;
2. Gain insight into the influence of some components of the algorithm.

For the first objective, we present the results of the DACCO optimization for all the aforementioned instances, and we analyze its performance based on two criteria:

1. Ability to discover the known optima;
2. Mean Best Fitness (MBF) and deviation from the optimum;

The first criterion is a widely adopted performance measure in cluster geometry optimization. Hence, for all instances, we show its success rate(number of

Parameter	**Value**
Runs	30
Population size	Equal to the number of atoms N
Number of evaluations	5000000
Morse Potential range (β)	14.0
Cell size (W)	0.6
Number of Atoms (N)	Between 30 and 50
Influence of the pheromones (α)	4
Continuous Local Search Iterations	1000
Continuous Local Search Accuracy	1.0E-8
Pheromone propagation value (p)	0.5
Neighborhood type	Moore Neighborhood
Neighborhood radius (r)	3
Discrete Local Search Iteration	10

Table 5.1: Parameter setting used in the experiments

times that it found the best-known solution) of the algorithm. In case it does not find the best-known solution we present the best solution found by the algorithm. The second criterion aims to complement our study, as it will provide information about the ability of DACCO to converge to promising areas of the search space.
Later, to assess the absolute performance of the DACCO, we compare its results with those achieved by two algorithms: a steady-state EA described in Pereira et al. [29], and the Particle Swarm Optimization algorithm (PSO) described in Lourenço et al. [24].

Experimental Settings

Table 5.1 lists the general parameters used in all of the experiments. We performed a total of 30 runs to make possible a statistical analysis. In each run we allowed our algorithm to perform 5000000 evaluations. It is important to refer that each iteration made by L-BFGS procedure counts as one evaluation. The size of the population depends on the size of the cluster being optimized. This parameter was set based on the reviewed literature, and some preliminary tests. The cell size (W) corresponds to size of each cell in our search space. The neighborhood used is the Moore neighborhood with $r = 3$. The number of iterations in the discrete local search is 10. The α value was set to 4, following the reviewed literature, and after the analysis of some preliminary results.

Statistical Analysis

While comparing our algorithm with the EA and the PSO we performed a statistical analysis to validate the results. We assumed that our data did not followed any distribution. Thus we applied the Mann-Whitney non-parametric test, at a 0.05 level of significance, to assess the statistical differences of the means, over the 30 runs of each of pair DACCO-EA and DACCO-PSO of algorithms. Furthermore the same test is used to perform a detailed analysis in the components of the algorithm. For multiple comparisons the p-value (0.05) used was adjusted using the Bonferroni correction method. To perform such correction we used used the following equation:

$$p_B = \frac{p}{nc} \tag{5.1}$$

where nc is the number of comparisons performed, and the p_B is the p-value to consider.

The hypothesis for comparing two algorithms were:

- $H_0 : u_1 = u_2$ - means of the algorithms were equal, as the null hypothesis.
- $H_1 : u_1 \neq u_2$ - means of the algorithms were not equal, as the alternative hypothesis.

Each time we applied a statistical test we looked for the results and concluded: if the p-value of the statistical test was smaller than the level of confidence, there was evidence to reject the null hypothesis H_0, and we could accept the alternative H_1. This means that there was evidence that the means were significantly different at the significance level of 0.05. In contrast, there was not enough evidence to reject H_0, and being so, we concluded that the means were not significantly different.

In the tables where we make the statistical analysis we use the following notation: "$>$" when the means are statistical different, and the first algorithm presents an higher mean value than the second, "$<$" when the means are statistical different, and the second algorithm presents an higher mean value than the first, and "$\sim$" when there is no statistical differences in the means.

5.2 DACCO: Experimental Results

In Table 5.2 we present the optimization results of short-ranged Morse Clusters between 30 and 50 obtained by the DACCO algorithm. The first column,

Instance, identifies the number of atoms of each cluster. The second column, *Optimum*, displays the potential energy of the best-known solution. The third column, *Best Solution Found*, displays the potential energy of the best solution found by the DACCO. The next two columns present the success rate (column *SR*) and the Mean Best Fitness (column *MBF*). The last column measures of how the MBF deviates, in percentage, from the best-known solution.

Instance	**Optimum**	**Best Solution Found**	**SR**	**MBF**	**Deviation**
30	-106.835790	-106.835790	28 / 30	-106.831095	0.004
31	-111.760670	-111.760670	30 / 30	-111.760670	0.000
32	-115.767561	-115.767561	30 / 30	-115.767561	0.000
33	-120.741345	-120.741345	30 / 30	-120.741345	0.000
34	-124.748271	-124.748271	30 / 30	-124.748271	0.000
35	-129.737360	-129.737360	30 / 30	-129.737360	0.000
36	-133.744666	-133.744666	30 / 30	-133.744666	0.000
37	-138.708582	-138.708582	28 / 30	-138.682731	0.019
38	-144.321054	-144.321054	25 / 30	-144.053535	0.185
39	-148.327400	-148.327400	26 / 30	-148.243303	0.057
40	-152.333745	-152.333745	25 / 30	-152.228797	0.069
41	-156.633479	-156.633479	11 / 30	-156.483040	0.096
42	-160.641020	-160.641020	5 / 30	-160.449243	0.119
43	-165.634973	-165.634973	6 / 30	-165.361457	0.165
44	-169.642441	-169.642441	3 / 30	-169.383463	0.153
45	-174.511632	-174.511632	3 / 30	-174.295931	0.124
46	-178.519320	-178.519320	2 / 30	-178.371855	0.083
47	-183.508227	-183.411312	0 / 30	-183.095976	0.225
48	-188.888965	-188.888965	19 / 30	-188.402414	0.258
49	-192.898412	-192.898412	15 / 30	-192.675230	0.116
50	-198.455632	-198.455632	12 / 30	-197.853115	0.304

Table 5.2: Optimization results of Morse Clusters between 30 and 50 obtained by DACCO

The results from Table 5.2 are encouraging, since the proposed approach was able to find almost all the best-known solutions for short-ranged Morse clusters between 30 and 50, missing only 1 instance. To the best of our knowledge, this is the first work to apply ACO algorithm to the problem of cluster geometry optimization, hence its results may play an important role in future proposals that intend to use the same approach.

By analyzing the success rate of the algorithm we noticed that its variation is irregular: whereas between 30 and 40 atoms the success rate is high, for clusters between 41 and 50 atoms the algorithm has more difficulties to find the optimum. This can in part be explained, by the fact the we keep the number of evaluation fixed for all the clusters. With more atoms, we will get a larger search space, and it is not surprising that the performance of the algorithm declines.

Looking at the MBF values, they are always close to the optimum, as the deviation values range between 0.000 % and 0.304 %. This result is interesting, as it shows that even if DACCO converges to local optima, it can accurately find low potential structures.

Another interesting aspect is that DACCO has a good performance while optimizing the so called "magic instances" of 30 and 38 atoms [13]. These instances define particularly rugged landscapes, and the majority of the unbiased algorithms tend to converge to local optima [13, 16].

5.3 Algorithm Comparison

In this section we compare our approach with others described in the literature. A direct comparison with other ACO approaches its not possible, due to absence of such approaches. Hence, we compare our approach with one of the Swarm Intelligence algorithms family, and with an another of the Evolutionary Algorithms family.

5.3.1 DACCO versus PSO

In Table 5.3 we compare the DACCO and PSO algorithms. The third and fourth column represent the success rate (SR) and the MBF of DACCO. The last two columns represent the SR and the the MBF of PSO.

The number of evaluations granted to the PSO algorithm were the same of the DACCO.

Looking at the results, we can conclude that the DACCO is better than PSO. We can see that, for almost all instances the SR of DACCO is higher than the SR of the PSO. It misses only the 47 atoms instance. Furthermore, analyzing the MBF values the DACCO presents the better values for all instances. These MBF results are interesting because they show that the DACCO algorithm can

		DACCO		PSO	
Instance	**Optimum**	**SR**	**MBF**	**SR**	**MBF**
30	-106.835790	28 / 30	-106.831095	4 / 30	-106.718221
31	-111.760670	30 / 30	-111.760670	19 / 30	-111.630904
32	-115.767561	30 / 30	-115.767561	20 / 30	-115.686360
33	-120.741345	30 / 30	-120.741345	19 / 30	-120.690258
34	-124.748271	30 / 30	-124.748271	15 / 30	-124.605299
35	-129.737360	30 / 30	-129.737360	6 / 30	-129.078905
36	-133.744666	30 / 30	-133.744666	14 / 30	-133.494812
37	-138.708582	28 / 30	-138.682731	12 / 30	-138.147442
38	-144.321054	25 / 30	-144.053535	8 / 30	-142.545537
39	-148.327400	26 / 30	-148.243303	7 / 30	-147.361971
40	-152.333745	25 / 30	-152.228797	7 / 30	-151.516586
41	-156.633479	11 / 30	-156.483040	2 / 30	-155.898689
42	-160.641020	5 / 30	-160.449243	4 / 30	-160.027062
43	-165.634973	6 / 30	-165.361457	4 / 30	-164.649840
44	-169.642441	3 / 30	-169.383463	3/ 30	-168.908517
45	-174.511632	3 / 30	-174.295931	3 / 30	-173.160552
46	-178.519320	2 / 30	-178.371855	1 / 30	-177.513539
47	-183.508227	0 / 30	-183.095976	1 / 30	-182.081130
48	-188.888965	19 / 30	-188.402414	2 / 30	-186.782038
49	-192.898412	15 / 30	-192.675230	4 / 30	-191.496032
50	-198.455632	12 / 30	-197.853115	1 / 30	-195.816027

Table 5.3: Experimental results of Morse cluster between 30 and 50 atoms obtained by the DACCO algorithm and the PSO

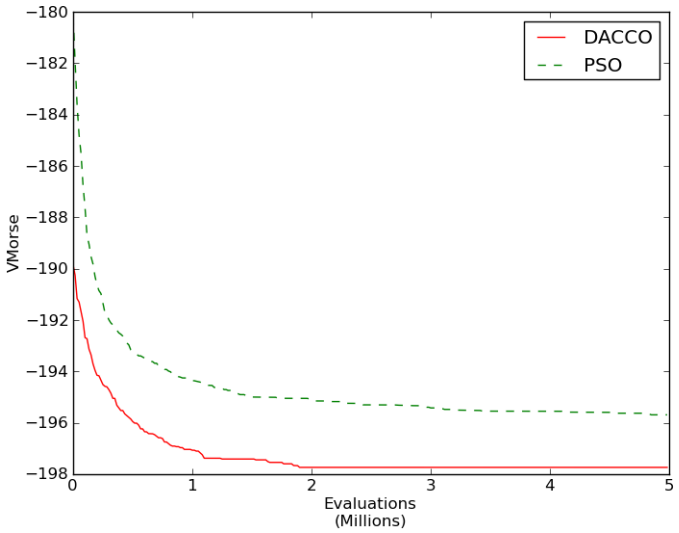

Figure 5.1: Evolution of the MBF of DACCO and PSO. The results were obtained with the Morse cluster of 50 atoms.

find solutions with better quality than the ones of the PSO. To confirm this, in Fig. 5.1 we present the evolution of the MBF for DACCO and PSO. The results are from the Morse cluster with 50 atoms, but the same trend is visible for other instances. The information in the chart shows that the MBF of the DACCO is lower during the whole optimization process.
Looking at Table 5.4, we can see that the means of DACCO are statistical different and they have an higher value. Hence we can concluded that DACCO performs better than the PSO.

Instance	**DACCO-PSO**
30	>
31	>
32	>
33	>
34	>
35	>
36	>
37	>
38	>
39	>
40	>
41	>
42	>
43	>
44	>
45	>
46	>
47	>
48	>
49	>
50	>

Table 5.4: Statistical results of comparing DACCO and PSO

5.3.2 DACCO versus EA

In Table 5.5 we compare the DACCO and EA algorithms. Like in the previous section, the third and fourth column represent SR and the MBF of DACCO. The last two columns represent the SR and the the MBF of EA. Again, the number of evaluation granted to the EA, was the same of the DACCO. An

		DACCO		EA	
Instance	**Optimum**	**SR**	**MBF**	**SR**	**MBF**
30	-106.835790	28 / 30	-106.831095	22 / 30	-106.794747
31	-111.760670	30 / 30	-111.760670	30 / 30	-111.760670
32	-115.767561	30 / 30	-115.767561	29 / 30	-115.766686
33	-120.741345	30 / 30	-120.741345	28 / 30	-120.697611
34	-124.748271	30 / 30	-124.748271	28 / 30	-124.715475
35	-129.737360	30 / 30	-129.737360	27 / 30	-129.623232
36	-133.744666	30 / 30	-133.744666	28 / 30	-133.715154
37	-138.708582	28 / 30	-138.682731	25 / 30	-138.610585
38	-144.321054	25 / 30	-144.053535	8 / 30	-143.130422
39	-148.327400	26 / 30	-148.243303	14 / 30	-147.958055
40	-152.333745	25 / 30	-152.228797	9 / 30	-151.886095
41	-156.633479	11 / 30	-156.483040	15 / 30	-156.547928
42	-160.641020	5 / 30	-160.449243	12 / 30	-160.518149
43	-165.634973	6 / 30	-165.361457	14 / 30	-165.254805
44	-169.642441	3 / 30	-169.383463	7 / 30	-169.303639
45	-174.511632	3 / 30	-174.295931	5 / 30	-174.102119
46	-178.519320	2 / 30	-178.371855	9 / 30	-178.389713
47	-183.508227	0 / 30	-183.095976	2 / 30	-183.153610
48	-188.888965	19 / 30	-188.402414	14 / 30	-188.160694
49	-192.898412	15 / 30	-192.675230	18 / 30	-192.627890
50	-198.455632	12 / 30	-197.853115	5 / 30	-197.688978

Table 5.5: Experimental results of Morse cluster between 30 and 50 atoms obtained by the DACCO algorithm and the EA

overview of the results presented in Table 5.5 reveal that DACCO has as good results as those of EA for the considered Morse clusters. Moreover, looking at the MBF values of both algorithms, we can see that they are very close, revealing that the solutions found by DACCO are as good as the ones found by the EA. To confirm this, in Fig. 5.2 we present the evolution of the MBF for DACCO and the EA. The results are from the Morse cluster with 50 atoms, but the same trend is visible for other instances. By observing it we see that

the MBF of DACCO tends to converge more rapidly than the MBF of the EA. However, at the end of the optimization process the MBF of DACCO and of EA tend to get equal.
Table 5.6 shows that DACCO is as effective as the EA.

Instance	**DACCO-EA**
30	$>$
31	$\simeq$
32	$\simeq$
33	$\simeq$
34	$\simeq$
35	$\simeq$
36	$\simeq$
37	$\simeq$
38	$>$
39	$>$
40	$>$
41	$\simeq$
42	$\simeq$
43	$\simeq$
44	$\simeq$
45	$\simeq$
46	$\simeq$
47	$\simeq$
48	$\simeq$
49	$\simeq$
50	$\simeq$

Table 5.6: Statistical results of comparing DACCO and EA

It is important to note that in four Morse instances our approach has statistical differences in the means. Some of these Morse instances, the ones with 30 and 38 atoms, are particularly hard to optimize, as they define very roughed landscapes, with many local minima [13].

Moreover, these results show that DACCO can compete with the EA, while optimizing Morse clusters with a number of atoms between 30 and 50. This is an important outcome since the EA used for comparison is a state-of-art method, and the algorithm proposed in this dissertation performs as good as it.

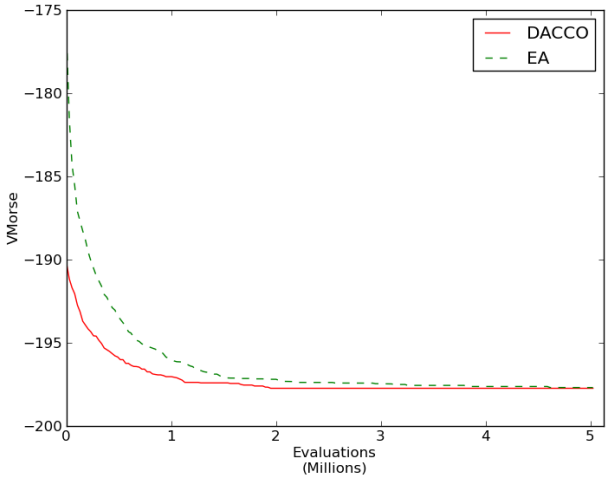

Figure 5.2: Evolution of the MBF of DACCO and EA. The results were obtained with the Morse cluster of 50 atoms.

5.4 Detailed Analysis

DACCO contains some components that enhance its effectiveness in cluster geometry optimization problems. In this section we present a set of additional tests to clarify the importance of some of these components. We selected clusters with size $N = \{30, 38, 45, 50\}$, and changes range from simple parameter setting variation to structural modifications in the DACCO framework. All comparisons in this work are made between the final DACCO version. These comparisons of results will help to clarify the influence of the different components used.

5.4.1 Cell Size

In this section we present the results of the variation of the cell size W. When we have a big W our search space will be smaller, but we will start to have more than on atom in one cell. Furthermore we have to be careful while defining the cell size, because we need to guarantee that our cube can host an entire cluster. Taking this into account it is important to assess how the algorithm behaves with different cell sizes. Table 5.7 present results obtained with additional setting $W = \{0.5, 0.6, 0.7\}$. A inspection of this table reveals that the outcomes of the optimization, with different W are similar. There is

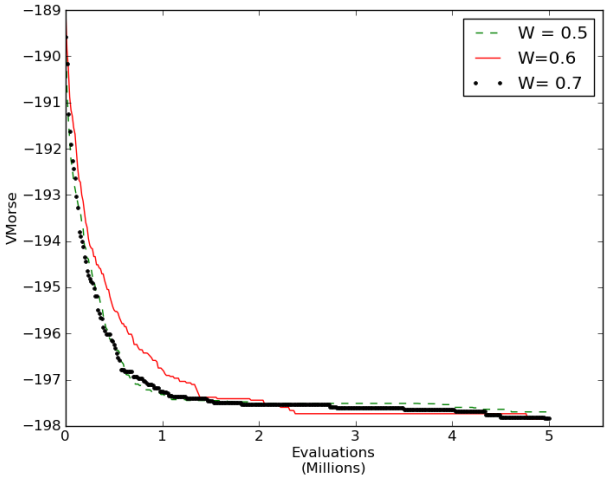

Figure 5.3: Evolution of the MBF of DACCO with different values of W. Results were obtained with the Morse instance with 50 atoms.

a slight trend for the $W = 0.6$ to obtain results of higher quality in both terms of success rate and MBF (Fig. 5.3). We performed a statistical analysis to confirm that there is no statistical evidence to which one of the W values is better.

Instance	$W = 0.5$		$W = 0.6$		$W = 0.7$	
	SR	MBF	SR	MBF	SR	MBF
30	28 / 30	-106.831095	28 / 30	-106.831095	30 / 30	-106.835790
38	20 / 30	-143.786016	25 / 30	-144.053535	19 / 30	-143.732512
45	1 / 30	-174.304756	3 / 30	-174.295931	2 / 30	-174.331536
50	4 / 30	-197.694748	12 / 30	-197.853115	9 / 30	-197.831498

Table 5.7: Optimization results obtained by DACCO with different values of W in the selected Morse cluster instances

5.4.2 Neighborhoods

In this section we compare two neighborhoods: the Moore Neighborhood and the Full Moore Neighborhood. We used the same $r = 3$ for both neighborhoods.

Table 5.8 present the results for both neighborhoods. Looking to the outcomes, we can see that for small instances the two neighborhoods have similar results.

However, looking for the results obtained in larger instances, the Moore Neighborhood has better results. This may be explained by the fact that in the Full Moore neighborhood we are taking into account much more cells, and thus, the cell selection process can become to greedy.
Table 5.9 shows that for the bigger instances, the algorithm using the Moore neighborhood has statistical differences in the means when compared to the Full Moore neighborhood. Fig. 5.4 presents the evolution of the MBF, and confirms that the Moore neighborhood achieves better results.

Instance	**Moore**		**Full Moore**	
	SR	MBF	SR	MBF
30	28 / 30	-106.831095	25 / 30	-106.823627
38	25 / 30	-144.053535	23 / 30	-143.942889
45	3 / 30	-174.295931	1 / 30	-173.998727
50	12 / 30	-197.853115	3 / 30	-197.001784

Table 5.8: Optimization results obtained by DACCO with different Neighborhoods in the selected Morse cluster instances

Instance	**Moore - Full Moore**
30	$\sim$
38	$\sim$
45	$>$
50	$>$

Table 5.9: Statistical results of comparing the different neighborhoods

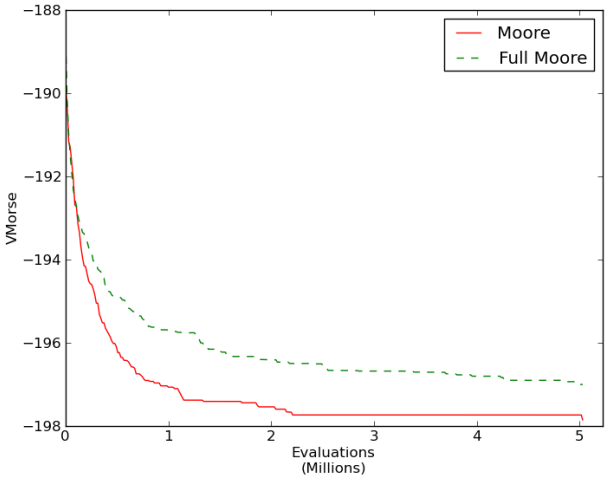

Figure 5.4: Evolution of the MBF of DACCO with different Neighborhoods. Results were obtained with the Morse instance with 50 atoms.

5.4.3 Neighborhood Radius

In this section we compare the radius of the two neighborhoods. Using the Moore Neighborhood, we tried experimented values of $r = \{1, 2, 3, 4\}$ while optimizing the subset of instances.

Table 5.10 present the results of the optimization. Looking at the results, we can observe that choosing a small radius ($r = 1$), the optimization process becomes too greedy, in a way that it always tries to build compact clusters. This works well for small instances, but for the large ones, we need some more freedom to place the atoms. However, with a large radius the optimization process starts to behave like the Full Moore Neighborhood.

Looking at these results, it is clear that we need to have a radius that is neither to small that it only looks at the closest positions, neither too large, that it starts destroying the cohesion of the cluster.

In Table 5.11 we present a statistical analysis of the radius used for experimentation. It clearly confirms that small radius do not help the search.

Fig. 5.5 presents the evolution of the MBF for the different values of r. These figures confirms that, with low values of r, the algorithm converges to local optima of inferior quality when compared, for example, with $r = 2$ or $r = 3$. Moreover, this chart shows that with $r = 4$ the MBF is smaller than the MBF of the version that uses $r = 3$

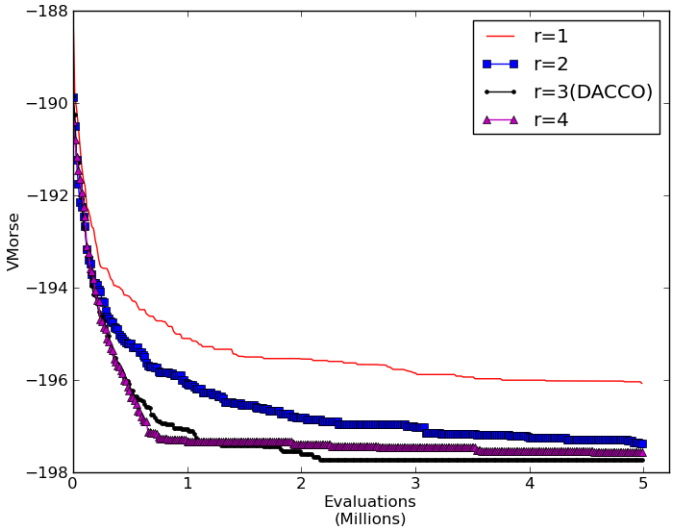

Figure 5.5: Evolution of the MBF of DACCO with different values of r. Results were obtained with the Morse instance with 50 atoms.

Instance	**r = 1**		**r = 2**		**r = 3**		**r = 4**	
	SR	MBF	SR	MBF	SR	MBF	SR	MBF
30	22 / 30	-106.816158	27 / 30	-106.828748	28 / 30	-106.831095	27 / 30	-106.828748
38	28 / 30	-144.211960	25 / 30	-144.052332	25 / 30	-144.053535	23 / 30	-143.946527
45	0 / 30	-173.593068	4 / 30	-174.338129	3 / 30	-174.295931	1 / 30	-174.338283
50	0 / 30	-196.105401	7 / 30	-197.387273	12 / 30	-197.853115	4 / 30	-197.605420

Table 5.10: Optimization results obtained by DACCO with different values of r in the selected Morse cluster instances

Instance	**r = 1 - r = 2**	**r = 1 - r = 3**	**r = 1 - r = 4**	**r = 2 - r = 3**	**r = 2 - r = 4**	**r = 3 - r = 4**
30	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
38	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$
45	$<$	$<$	$<$	$\sim$	$\sim$	$\sim$
50	$<$	$<$	$<$	$\sim$	$\sim$	$\sim$

Table 5.11: Statistical results of comparing the neighborhood radius

5.4.4 Discrete Local Search

In this section we analyze the DACCO algorithm with and without the discrete local search procedure.

The discrete local search procedure was designed to cluster geometry optimization problems, and with this set of experiments we intend to show its importance in the optimization process. Table 5.12 shows that without a local search procedure in the discrete space, the algorithm has a poor performance. These results show that the discrete local search is a vital component to the success of the algorithm.

Table 5.13 presents a statistical analysis that confirms the importance of the discrete local search. Fig. 5.6 shows the evolution of the MBF in the optimization process. It clearly depicts that the MBF without discrete local search, is inferior to the one with discrete local search.

Instance	**DACCO**		**DACCO without Discrete Local Search**	
	SR	MBF	SR	MBF
30	28 / 30	-106.831095	2 / 30	-106.406177
38	25 / 30	-144.053535	1 / 30	-142.077711
45	3 / 30	-174.295931	0 / 30	-172.596377
50	12 / 30	-197.853115	2 / 30	-194.567170

Table 5.12: Optimization results obtained by DACCO with and without Discrete Local Search in the selected Morse cluster instances

Instance	**DACCO - DACCO without Discrete Local Search**
30	>
38	>
45	>
50	>

Table 5.13: Statistical results of comparing DACCO and DACCO without local search

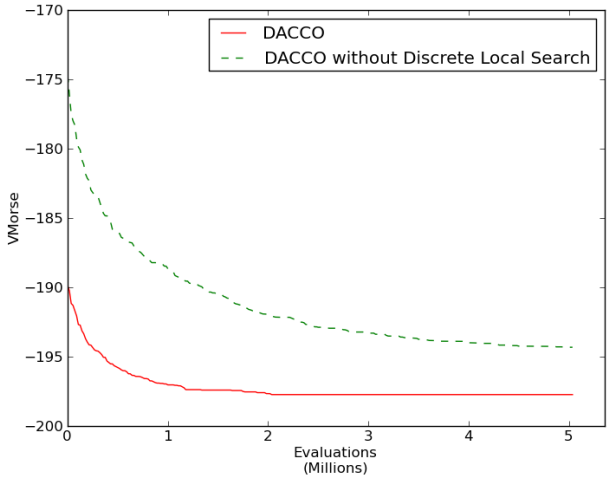

Figure 5.6: Evolution of the MBF of DACCO with and without Discrete Local Search. Results were obtained with the Morse instance with 50 atoms.

5.4.5 Pheromone Propagation

In this section we present the results of having a mechanism of pheromone propagation. This mechanism allows a smoother distribution of the pheromones in the search space (see Fig 5.7 and Fig.5.8).
Looking at Table 5.14, we can see that the results with and without pheromone propagation are similar. Furthermore, we performed a statistical analysis and confirmed that there was no statistical evidence to choose one of the methods.

Instance	**DACCO**		**DACCO without pheromone propagation**	
	SR	MBF	SR	MBF
30	28 / 30	-106.831095	25 / 30	-106.824054
38	25 / 30	-144.053535	22 / 30	-143.893024
45	3 / 30	-174.295931	2 / 30	-174.311589
50	12 / 30	-197.853115	10 / 30	-197.652445

Table 5.14: Optimization results obtained by DACCO with and without Discrete Local Search in the selected Morse cluster instances

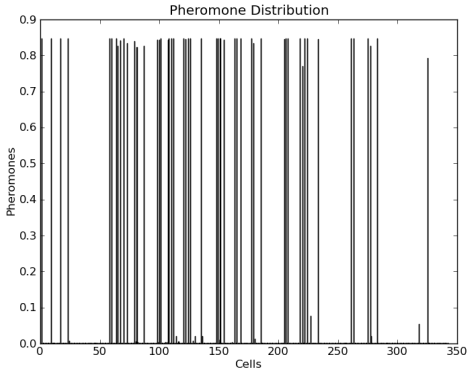

Figure 5.7: Pheromone distribution without pheromone propagation in the final iteration of the optimization of a Morse cluster with 50 atoms.

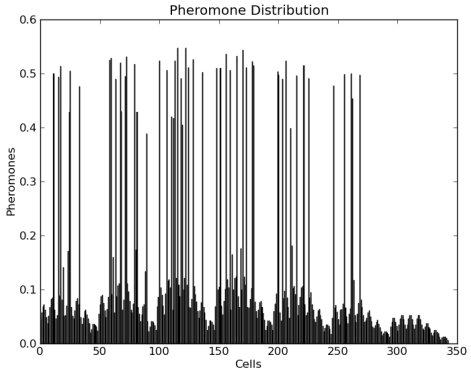

Figure 5.8: Pheromone distribution with pheromone propagation in the final iteration of the optimization of a Morse cluster with 50 atoms.

Chapter 6

Conclusion

The main goal of this dissertation is to propose a new algorithm to tackle cluster geometry optimization problems. The proposed algorithm is based on swarm intelligence, more specifically in a discrete variant of the Ant Colony Optimization (ACO). In this dissertation we proposed some new components to improve the effectiveness of the algorithm:

- Discretization of the search space. Cluster geometry optimization problems are continuous, and it was necessary to build a method to discretize the problem so that a discrete version of ACO could be applied;
- Several methods to find feasible neighborhoods were tested. With these methods we intended to introduce more information about the problem while building the feasible neighborhoods;
- Application of a continuous local optimization method combined with the ACO algorithm;
- The mapping between solutions in the continuous and discrete spaces. With the mapping used we tried to keep as much information as possible between spaces;
- The proposal of a discrete local search method, that ought to be crucial in the achievement of good results.

Furthermore we studied the state-of-art in both ACO algorithms and in optimization methods used to tackle the problem of cluster geometry optimization.

The proposed algorithm, DACCO, was tested in several instances of short-ranged Morse clusters. Its performance was compared with two other approaches applied to the same problem instances: a Particle Swarm Optimiza-

tion (PSO) algorithm and an Evolutionary Algorithm (EA). We concluded that DACCO performed better than the PSO in all the tested instances. When compared to the EA, DACCO obtained similar results, having a better performance in four of the tested Morse instances.

We performed a detailed analysis in some of the proposed components, in order to assess their importance in the algorithm.

6.1 Future Work

In this type of research there is always work to do. The first important aspect that should be addressed is the application of the DACCO algorithm to bigger instances of the short-ranged Morse clusters. Another aspect that should be study is the application of heuristics to improve the optimization process. The study and implementation of new mechanisms to find feasible neighborhoods, should be of great interest, once this is one of the key aspects of the ACO algorithms. The improvement of the discrete local search method should be addressed as well, since it was proven that this component is vital to DACCO succeed in the optimization process. The comparison of our results with other approaches that are present in the literature is another aspect that should be made in the near future in order to assess its effectiveness. Finally, study the possible application of DACCO to the molecular cluster optimization. Currently there are no effective crossover operators for dealing with cluster that are composed by molecules. Since ACO algorithms do not rely on these type of operators, they might be a relevant method to tackle this optimization problem.

Bibliography

[1] G. Bilchev and I. Parmee. The ant colony metaphor for searching continuous design spaces. In Terence Fogarty, editor, *Evolutionary Computing*, volume 993 of *Lecture Notes in Computer Science*, pages 25–39. Springer Berlin / Heidelberg, 1995.

[2] C. Blum and M. Dorigo. The hyper-cube framework for ant colony optimization. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(2):1161–1172, 2004.

[3] P. A. Braier, R. S. Berry, and D. J. Wales. How the range of pair interactions governs features of multidimensional potentials. *The Journal of Chemical Physics*, 93(12):8745–8756, 1990.

[4] B. Bullnheimer, R. F. Hartl, and C. Strauß. A new rank based version of the ant system - a computational study. *Central European Journal for Operations Research and Economics*, 7:25–38, 1997.

[5] L. Cheng and J. Yang. Global minimum structures of morse clusters as a function of the range of the potential: $81 \leq n \leq 160$. *The Journal of Physical Chemistry A*, 111(24):5287–5293, 2007.

[6] D. M. Deaven and K. M. Ho. Molecular geometry optimization with a genetic algorithm. *Physical Review Letters*, 75(2):288–291, 1995.

[7] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[8] M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[9] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26(1):29–41, 1996.

[10] M. Dorigo and T. Stützle. *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA, 2004.

[11] J. P. K. Doye, R. H. Leary, M. Locatelli, and F. Schoen. Global optimization of morse clusters by potential energy transformations. *Informs Journal on Computing*, 16(4):371–379, 2004.

[12] J. P. K. Doye and D. J. Wales. Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116, 1997.

[13] J. P. K. Doye and D. J. Wales. Structural consequences of the range of the interatomic potential: a menagerie of clusters. *Journal Chemical Society, Faraday Transactions*, 93:4233–4243, 1997.

[14] J. P. K. Doye and D. J. Wales. Thermodynamics of global optimization. *Physical Review Letters*, 80(7):1357–1360, 1998.

[15] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[16] A. Grosso, M. Locatelli, and F. Schoen. A population-based approach for hard global optimization problems based on dissimilarity measures. *Mathematical Programming*, 110:373–404, 2007.

[17] R. L. Johnston. Evolving better nanoparticles: Genetic algorithms for optimising cluster geometries. *Dalton Transactions*, pages 4193–4207, 2003.

[18] J. Kennedy, Russell C. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann, 2001.

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

[20] M. Kong and P. Tian. A direct application of ant colony optimization to function optimization problem in continuous domain. In Marco Dorigo, Luca Gambardella, Mauro Birattari, Alcherio Martinoli, Riccardo Poli, and Thomas Stützle, editors, *Ant Colony Optimization and Swarm Intelligence*, volume 4150 of *Lecture Notes in Computer Science*, pages 324–331. Springer Berlin / Heidelberg, 2006.

[21] J. E. Lennard-Jones. Cohesion. *Proceedings of the Physical Society*, 43(5):461, 1931.

[22] Z. Li and H. Scheraga. Monte Carlo-minimization approach to the multiple-minima problem in protein folding. *Proceedings of the National*

Academy of Sciences of the United States of America, 84(19):6611–6615, 1987.

[23] C. D. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[24] N. Lourenço and F. Pereira. PSO-CGO: A Particle Swarm Algorithm for Cluster Geometry Optimization. *International Journal of Natural Computing Research*, 2(1):1–20, 2011.

[25] S. Luke. *Essentials of Metaheuristics*. 2009. available at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[26] V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *Informs Journal on Computing*, 11:358–369, 1999.

[27] L. Melo, F. Pereira, and E. Costa. MC-ANT: A multi-colony ant algorithm. In Pierre Collet, Nicolas Monmarché, Pierrick Legrand, Marc Schoenauer, and Evelyne Lutton, editors, *Artifical Evolution*, volume 5975 of *Lecture Notes in Computer Science*, pages 25–36. Springer Berlin / Heidelberg, 2010.

[28] P. M. Morse. Diatomic molecules according to the wave mechanics. ii. vibrational levels. *Physical Review*, 34(1):57–64, 1929.

[29] F. Pereira and J. Marques. A study on diversity for cluster geometry optimization. *Evolutionary Intelligence*, 2:121–140, 2009.

[30] C. Roberts, R. L. Johnston, and N. T. Wilson. A genetic algorithm for the structural optimization of morse clusters. *Theoretical Chemistry Accounts: Theory, Computation, and Modeling (Theoretica Chimica Acta)*, 104:123–130, 2000.

[31] X. Shao, L. Cheng, and W. Cai. A dynamic lattice searching method for fast optimization of lennard–jones clusters. *Journal of Computational Chemistry*, 25(14):1693–1698, 2004.

[32] B. M. Smirnov, A. Y. Strizhev, and R. S. Berry. Structures of large morse clusters. *The Journal of Chemical Physics*, 110(15):7412–7420, 1999.

[33] K. Socha. ACO for continuous and mixed-variable optimization. In Marco Dorigo, Mauro Birattari, Christian Blum, Luca M.Gambardella, Francesco Mondada, and Thomas Stützle, editors, *Ant Colony, Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 53–61. Springer Berlin / Heidelberg, 2004.

[34] K. Socha and M. Dorigo. Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3):1155 – 1173, 2008.

[35] Frank H. Stillinger. Exponential multiplicity of inherent structures. *Physical Review E*, 59(1):48–51, 1999.

[36] T. Stützle and H. H. Hoos. $\mathcal{MAX}-\mathcal{MIN}$ Ant system. *Future Generation Computer Systems*, 16(8):889 – 914, 2000.

[37] S. Tsutsui. Ant colony optimisation for continuous domains with aggregation pheromones metaphor. In *Proceedings of the 5th International Conference on Recent Advances in Soft Computing*, pages 207–212, 2004.

[38] L. T. Wille and J Vennik. Computational complexity of the ground-state determination of atomic clusters. *Journal of Physics A: Mathematical and General*, 18(8):419–422, 1985.

[39] A. Zaslavski and J. P. K. Doye. Physical perspectives on the global optimization of atomic clusters. In Panos Pardalos and János D. Pintér, editors, *Global Optimization*, volume 85 of *Nonconvex Optimization and Its Applications*, pages 103–139. Springer US, 2006.

Appendix A

Implementation Details

Since this dissertation is part of the Masters Degree in Informatics Engineering, it is important to refer some technical aspects about the implementation. In this chapter we give a brief description of the technical choices made, and we present a quick overview of the entire system.

A.1 Technical Choices

To implement the first versions of the system we decide to use the Python language. One of the reasons for this is related with the simplicity of the language, and easiness in making changes. The other reason is related with the L-BFGS method, which is written in Fortran. With python it is easy to connect modules made in Fortran with our code.

After we have a stable version, we decided to write it in C. The reason for this was the performance of the python language.

The C language, despite being very fast, has some programming overhead, when compared, for instance, with python. During the conversion of the python code to C, some bugs appeared, as one should expect. They were solved using the *gdb* debugging system.

We used *Git* as revision control system. As online repository our choice was the *Github* platform, with private repositories.

A.2 Architecture Overview

We tried to keep our architecture simple and modular. In Fig.A.1 we present a class diagram, which in our opinion gives a better insight in how the system is organized.

DACCO – Main class that runs the ACO algorithm for a certain number of populations.

Population – Class that contains a certain number of individuals, with common properties.

Ant – Class that simulates the real ant. It is responsible for the individual methods of each ant. The main methods are:

- Construction of individual solutions:
  - Choose feasible neighborhoods;
  - Apply a probabilistic rule to choose which components should be added to the solutions.
- It calls the Fitness Function, in order to assess the quality of the solution that has been built;
- Mapping of the continuous solutions to the discrete space;
- Discrete local search. This local search works in the search space, and tries to improve the ant solutions applying random changes to the current solutions.

Utils – Class that has a set of procedures to help all the other classes accomplish their individual tasks. Examples of these methods are: the euclidean distance method, that receives to points in the space and determines their euclidean distance; the method to read the parameters file; methods to write data to files, in order to allow us to see what is happening in the optimization process.

Fitness Function – Class that is responsible for evaluate the quality of the individuals that belong to our system. This class receives an ant solution, that is already in the continuous space, and it performs the evaluation of the solutions. The evaluation is performed by a Fortran function. In order to use it, we applied two different techniques:

- To connect the Fortran code with C we created several object files and the we compiled them together. One aspect that we had to take into

account was that sometimes the Fortran functions have and "_" after the name.

- To connect Fortran with Python, we created a static library, using the Cython [1] framework.

[1] http://cython.org/

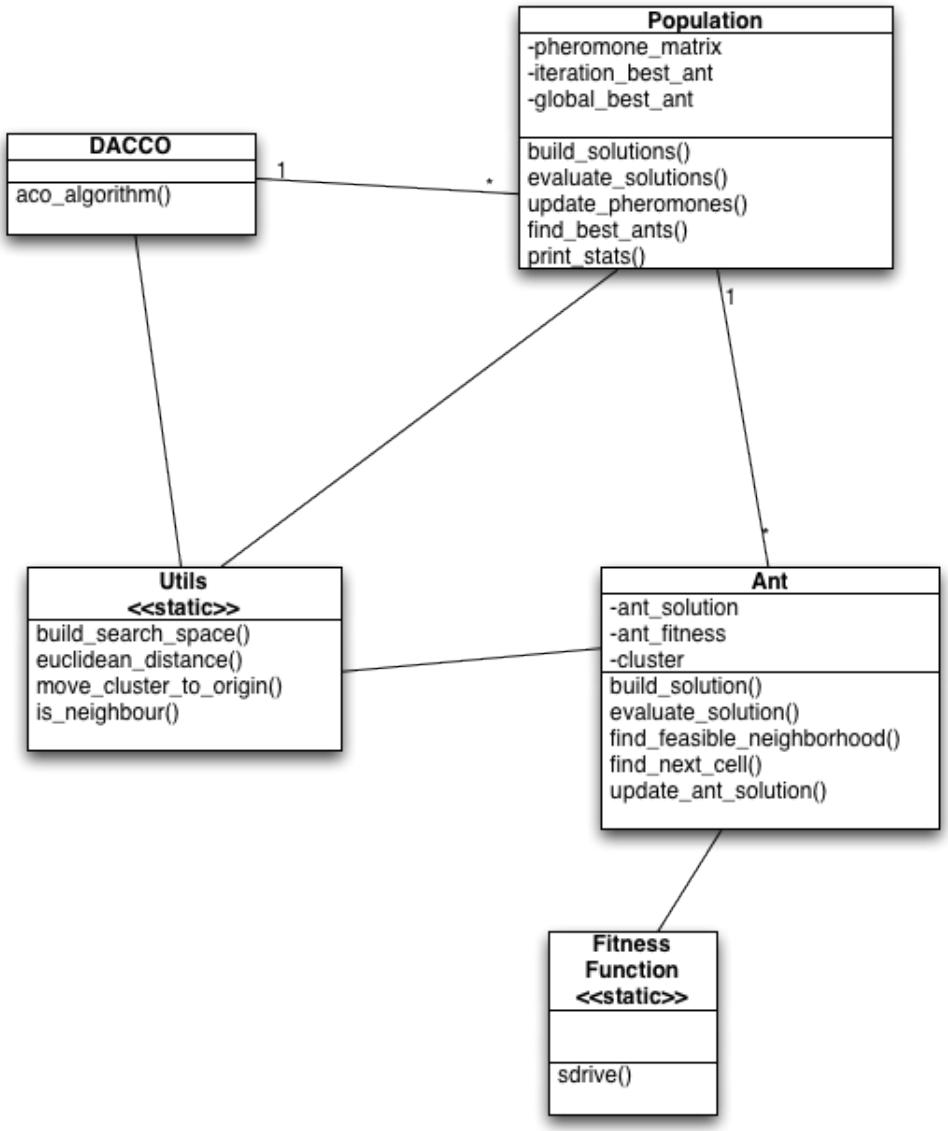

Figure A.1: CGACO Class Diagram