Icosahedral clusters

Towards understanding the nature of the formation of quasicrystals

by

R. Calegari

to obtain the degree of Bachelor of Science in Applied Mathematics
at the Delft University of Technology,
to be defended publicly on Wednesday July 19, 2023 at 10:30 AM.

Student number:	5149703	
Project duration:	March 1, 2023 – July 19, 2023	
Thesis committee:	Dr. D. Palagin,	TU Delft, supervisor
	Dr. C. Kraaikamp,	TU Delft

TU Delft

Contents

1

Introduction

Understanding the formation of quasicrystals is a complex task that requires the integration of computational tools and simulations to model the behavior of atomic clusters. This thesis aims to contribute to the current understanding of quasicrystal formation by combining global geometry optimization and atomic simulations using the Oscillating Pair Potential (OPP). These methods, although widely used separately, have not yet been combined to investigate the energetically stable structures that can form quasicrystalline materials and exhibit non-periodic long-range order.

Chapter 2 gives a foundation for the next chapters by introducing the topic of quasicrystals. It starts by going into the role of symmetries and order in the atomic structure of materials. It provides a brief explanation of the traditional understanding of crystals, where atoms are arranged in a periodic pattern governed by translational and rotational symmetries. Furthermore, it discusses the breakthrough discovery of icosahedral quasicrystals, which challenged the diffused belief of the presence of periodic order in all ordered materials. The presence of forbidden rotational symmetries and aperiodic long-range order in quasicrystals opened up new possibilities for unique material properties and applications.

Chapter 3 provides a detailed discussion of the computational tools necessary to set up simulations to model atomic clusters. It introduces interatomic potential formulas and their relationship with the position and potential energy of atoms. Two potential formulas are presented: the standard Lennard-Jones potential and the 3 wells Oscillating Pair Potential, specifically designed to reproduce the interactions in materials prone to form icosahedral quasicrystals. Additionally, the concept of Molecular Dynamics is introduced as a tool to represent the movement of atoms over time.

Chapter 4 focuses on investigating the energetically-favorable configurations that a system of particles governed by a certain potential can assume. It introduces the concept of the Potential Energy Surface (PES) and its relationship to stable cluster configurations. The chapter discusses different local optimization algorithms designed to find a stable equilibrium for a given initial configuration, and explains the use of local optimization within the global optimization algorithm, which aims to find the global minimum of the system, that is the most energetically-favourable configuration.

Chapter 5 presents diffraction patterns and Fourier Transforms, a mathematical tool widely used in the analysis of atomic structures in crystallography.

In Chapter 6, the theory discussed in the previous chapters is implemented using the HOOMD-blue Python package. The chapter presents the implementation of the methods, including the global optimization algorithm, and describes the setup of simulations for clusters with varying numbers of atoms using both the Lennard-Jones potential and the Oscillating Pair Potential. The results obtained from the simulations are analyzed, comparing the stable structures formed by the clusters governed by different potentials. Additionally, the chapter explores the possibility of obtaining icosahedral quasicrystalline structures starting from specific initial configurations.

Finally, Chapter 7 summarizes the findings of the project and provides recommendations to improve and expand the research. The combined use of global geometry optimization and atomic simulations with the OPP demonstrated the ability to identify energetically favorable configurations that exhibit a certain degree of order. However, the results need further validation using diffraction patterns, which in this project was not fully feasible due to the small size of the investigated clusters. The global optimization algorithm also requires improvement, particularly in handling memory management issues and obtaining a higher number of minima configurations for larger clusters. Recommendations for future studies include solving these aforementioned issues and testing bigger clusters to see if recognizable forbidden symmetries can be observed in diffraction patterns. Additionally, exploring the possibility of arranging atoms in pre-built blocks favorable for settling into long-range ordered structures could provide valuable insights.

By integrating these computational tools and geometry optimization techniques, this thesis contributes to the understanding of icosahedral quasicrystal formation and provides a foundation for further research in this field.

2

Why Icosahedral Quasicrystals?

Symmetries are abundant in the world and permeate reality up to its very core in the atomic structure of matter. For many years, solid state matter was thought to be either amorphous, that is completely disordered, or characterized by translational and specific rotational symmetries. This latter type of matter got the name of crystals. Crystals are a form of solid matter where atoms are arranged in an ordered periodic pattern. As Janot explains in his book about quasicrystals [1], in practice the atomic arrangement in matter is never perfect. But this aspect is neglected in crystallography and crystals are described by reference to perfect infinite arrays of geometrical points, called *lattices.* In a lattice, every point has identical surroundings; all lattice points are equivalent and the crystal lattice therefore exhibits perfect translational symmetry. Relative to any arbitrarily chosen origin, any other lattice point has the position vector:

$$\mathbf{r} = n_1\mathbf{a} + n_2\mathbf{b} + n_3\mathbf{c} \tag{2.1}$$

The volume associated with a single lattice point is unique, but there is a choice regarding the vectors **a**, **b**, **c**. The volume unit depicting the lattice is called the *unit cell.* A group of atoms can be associated with each lattice point. Then the position vector of an atom may be written:

$$\mathbf{r}_j = n_1\mathbf{a} + n_2\mathbf{b} + n_3\mathbf{c} + \mathbf{R}_j, \tag{2.2}$$

where $\mathbf{R}_j$ is the position vector within the unit cell relative to the lattice point (n_1, n_2, n_3). The group of atoms which is associated with every lattice point is called a basis. A crystal structure is therefore specified by its lattice and a basis. Geometrically, the introduction of the basis induces new symmetry elements such as rotations or reflections. Each symmetry operation or combination of operations must turn the atomic arrangement of the crystal into itself.

A symmetry operation is an operation that can be performed either physically or imaginatively that results in no change in the appearance of an object [2]. Looking in particular at rotational symmetries, if an object can be rotated about an axis and repeats itself every $360/n$ degrees of rotation, it is said to have an axis of n-fold rotational symmetry. The possible types of rotational symmetry in crystals are 2-fold, 3-fold, 4-fold, and 6-fold rotation (crystallographic restriction theorem [3]). On the other hand, geometries that exhibit symmetries such as 5-fold, 8-fold or 10-fold can't possibly occur in crystals; in fact, they are not periodic, i.e. it is not possible to arrange such geometries in a way that they completely fill up the space without overlapping.

(a) (b)

Figure 2.1: Squares and equilateral triangles repeat themselves upon a rotation of respectively 90° and 120°. They have a 4-fold and a 6-fold rotation axis, compatible with periodic order.

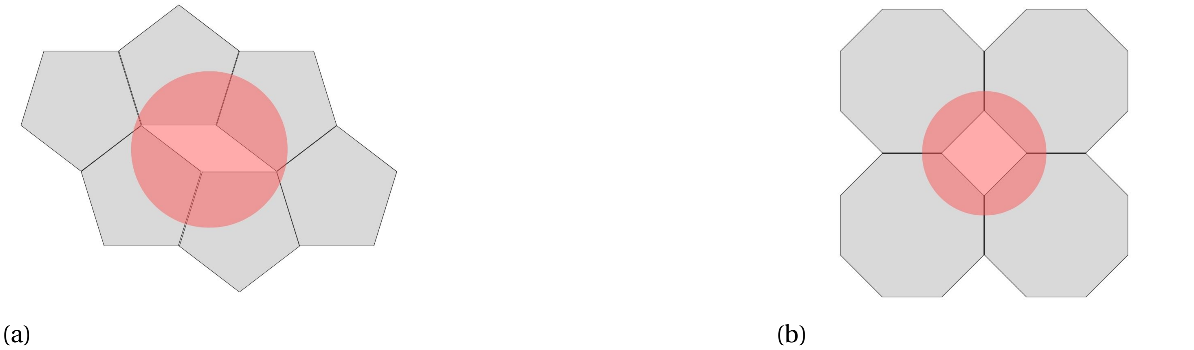
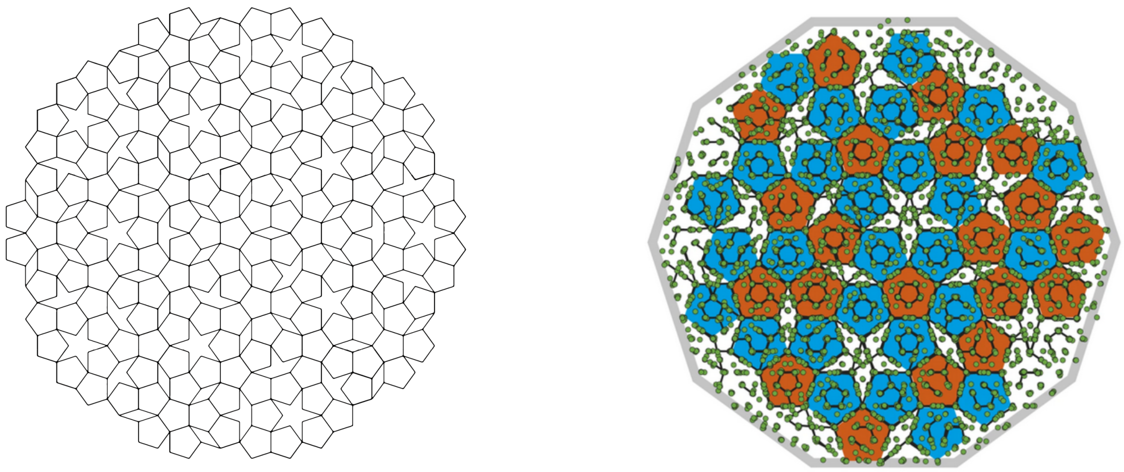

(a) (b)

Figure 2.2: Pentagons and octagons have a 5-fold and a 8-fold rotation axis. There is no way to combine them such that the fill the space without leaving any gaps. They are incompatible with periodic order.

Therefore, up to the end of the 1970s, it was believed that any solid was composed of microscopic building blocks with exclusively 2, 3, 4, or 6-fold symmetry. However, in those years, the physicist P. Steinhardt and the doctoral student D. Levine [4] began to investigate the possibility of materials with 5-fold symmetry, in which the atoms would have an ordered, though non-periodic, arrangement, by taking inspiration from the Penrose tilings [5], a well-known type of tessellation in which two shapes combine forming highly ordered but not periodic patterns; see an example in Figure 2.3.

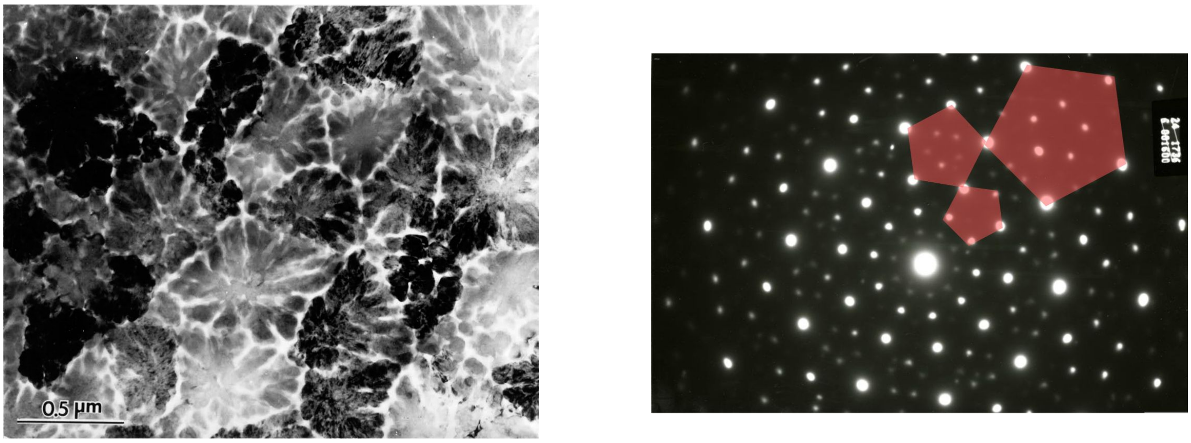


(a)

(b)

Figure 2.3: (a) illustrates the structure of the Penrose tiling P1, that consists of four shapes: a pentagon, a boat, a star and a rhombus. (b) is the projection of a cluster of atoms; the shapes overlapped on the image highlight that the particles are arranged into the P1 tiling. Taken from [6].

In 1984, the material scientist Dan Shechtman confirmed their claims and in a paper published in the prestigious physical journal *Physical Review Letters* [7], he stated to have found that the structure of a rapidly cooled alloy of aluminium and manganese had a 10-fold symmetry, which was in fact the first discovered quasicrystal [8]. The distinctive and surprising characteristic of quasicrystals, called quasiperiodicity, manifested in the lack of a continuous translational periodicity and the presence of rotational symmetries forbidden from the crystallographic restriction theorem, which cause the structure to have a long range aperiodic order.



(a) (b)

Figure 2.4: Al$_6$Mn alloy quasicrystal discovered by Dan Shechtman. (a) gives the first view of a crystal structure with icosahedral symmetry, and (b) shows its electron diffraction pattern. It can be clearly distinguished a 10-fold symmetry axis around the origin. Note the presence of equilateral pentagons in the structure. Taken from [9].

Since their existence had finally been accepted by the scientific community, more and more quasicrystal materials were created in the lab, and now hundreds of different quasicrystals can be counted with structures that exhibit 5-fold, 8-fold and 10-fold symmetries previously believed impossible. The findings are more than just conceptual. The first quasicrystal materials discovered were alloys usually involving aluminium with one or more other metals. The applications comprised first using them in non-stick frying pans for their high hardness and low conductivity [10], and later for making anti-corrosive coatings for surgical equipment and dental instruments [11] [12]. Quasicrystals have not been found only in metals, but in other materials such as polymers and recently also in materials made from self-assembling nanoparticles [13]. A new exciting field of research deals with the use of self-assembly processes to produce quasicrystals from different non-atom building blocks, including DNA and organometallic molecules [14] [15]. While the vast majority of quasicrystal currently discovered are human-made, there are some that have been found occurring in nature. After the physicist Steinhardt theorized the existence of quasicrystals, his research brought him to look for naturally formed quasicrystals and in 2009 he identified a naturally occurring quasicrystal structure in a speck of a rare mineral (later renamed icosahedrite after the geometry present in the atomic structure) from a meteorite landed in Chukhotka, Russia [16]. A particularly note-worthy type of quasicrystals is Icosahedral Quasicrystals (IQCs). Their importance comes from the abundance of icosahedral symmetry in nature and from the fact that, unlike axially symmetric quasicrystals as those based on decagonal symmetry which is periodic in one dimension, they are aperiodic in all dimensions and thus do not rely on any of the properties connected to the periodicity of a material in its atomic structure [6]. This type of quasicrystals has been experimentally observed only in intermetallic compounds, but the possibility of realizing IQCs patchy colloids using DNA particles has been hypothesized [17].

The wide application range for quasicrystal structures makes it very valuable to have a model to theoretically study the behaviour of such materials. The difference between crystal and quasicrystals at the atomic structure level causes the two materials to have different physical and chemical properties such as unconventional magnetic and conductivity behaviours. Thus, while we know how crystalline structures with a periodic arrangement form and behave, and modeling them is relatively simple thanks to the fact that knowing the unit cell we can describe the entirety of the crystal, quasicrystals are still relatively unknown, and their aperiodicity makes it a challenge to accurately model and simulate their assembly, without even talking about their growth. This is particularly true for IQCs: computational models of these structures are often built over simplifying assumptions, such as using short-range interactions or even hard sphere systems, where the potential energy vanishes unless the two spheres overlap, in which case it is infinite. These simplifying assumptions take the model far from conditions actually occurring in intermetallic compounds. A note-worthy breakthrough to this problem happened recently when Engel et al. [6] demonstrated that it is possible to robustly obtain the assembly of IQCs in a system consisting of identical particles interacting via an isotropic pair potential. While being a relatively simple formula, this potential accurately reproduces the interactions of atoms within alloys, materials where IQCs are formed.

In the next chapter the principles of atomic interactions will be explained and the ways to simulate their behavior will be presented.

3

Interatomic Interactions

This chapter will discuss the computational tools necessary to set up a simulation to model the behaviour of a cluster of atoms. In Section 3.1 the forces operating between atoms will be introduced, as well as the relation between the potential energy and the position of atoms in space; the discussion in this section will be largely based on the book *Interatomic Potentials* [18] by I. Torrens. Then two formulas will be presented: the simplest and most widely used potential formula, the Lennard-Jones potential (Section 3.1.1), and a specific potential formula for materials prone to form icosahedral quasicrystals, the 3w-OPP (Section 3.1.2). Finally, in Section 3.2 the concept of Molecular Dynamics will be presented, a tool to represent the movement of the atoms through time.

3.1. Interaction Potential

The way clusters of atoms assemble together is defined by the interaction forces between them. While atoms are usually considered and described as a unit ball, they have a composite nature, thus the interaction force between them is in fact resulting from the distinct interactions of their electrically charged components.

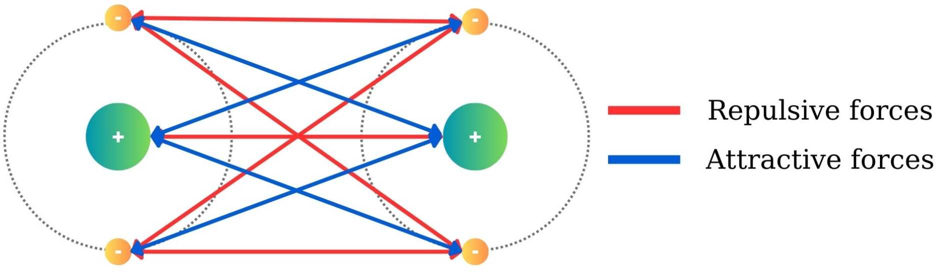


Figure 3.1: Illustration of the interatomic forces that occur between distinct subatomic components.

Consider two atoms made up of a nucleus and electrons orbiting around it. The presence of subnuclear components (protons and neutrons) is neglected since the forces acting between them are so much bigger than the interatomic forces that the nucleus can be considered a indivisible entity. The nucleus of one atom, having positive charge, has a significant influence on the cloud of outer electrons around the other atom resulting in attraction. At the same time the outer electrons around the first atom act on the cloud of electrons around the second, repelling it (van der Waals forces). The expression for the complete interatomic forces between two atoms becomes then a considerable complex many-body problem. Technically it is possible to formulate an accurate expression for these interaction forces. However, this would yield extremely complicated equations, except for the simplest systems. Although modern computing systems would be able to handle such heavy computations, it has become usual to represent the interaction forces with approximative formulas. These approximations are usually achieved by either simplifying theoretical formulas, or fitting a parametric formula to experimental data by tuning the parameters [18].

In general, the force between two atoms is expressed in terms of their potential energy of interaction, or interatomic potential [18]. As it was said for the forces, if we were to define the interaction potential between two atoms in a rigorous way, a quite complex system would take shape. Given two atoms with nuclear charges Z_1e and Z_2e, containing n electrons and divided by a distance r_{12}, the accurate definition of the potential energy is:

$$\Phi(r_{12}) = \frac{1}{2}\sum_{i\neq j=1}^{n}\frac{e^2}{r_{ij}} - \sum_{i=1}^{n}(\frac{Z_1e^2}{r_{i1}} + \frac{Z_2e^2}{r_{i2}}) + \frac{Z_1Z_2e^2}{r}[18] \tag{3.1}$$

The three terms in the equation respectively refer to the electron-electron repulsion, the electron-nucleus attraction, and the nuclear repulsion. Clearly it is desirable to reduce the problem. Generally the interaction potential is made depend exclusively on the distance r between the atoms and numerical parameters; this approach yields accurate enough results and is thus preferred. The relation between the force and the interatomic potential $V(r)$ is defined as follows:

$$F(r) = -\frac{\partial V(r)}{\partial r} \tag{3.2}$$

Since the interatomic potential ultimately depends not only on the relative distance of the particles but also on the nuclear charges and amount of electrons in the interacting atoms, the interatomic potential cannot be limited to a universal formula valid for any material. Therefore there is not a single one, but various potential formulas which have been researched and studied, each fit for a particular type of material or combination of atoms.

Rather than deducing the potential using theoretical considerations, it is usually more beneficial to derive it "empirically". So most of the potentials currently used are based on relatively simple expressions, not necessarily justified by the theory, which contain parameters that can be adjusted to fit the formula to experimental data.

3.1.1. (12-6) Lennard-Jones Potential

The (12-6) Lennard-Jones potential is a formula that gained big popularity since it was first proposed. It was first introduced to describe the interactions in inert gases, but it has been adopted in the modeling of many other materials, such as metals. While it doesn't provide the most accurate representation of the interactions in most of them, it has been widely used thanks to its simplicity. The Lennard-Jones potential is expressed as follows:

$$V(r) = \begin{cases} 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right], & \text{if } r \leq r_{\max} \\ 0 & \text{if } r > r_{\max} \end{cases} \tag{3.3}$$

where

- ϵ is the pair well depth. It describes how strongly the two atoms attract each other;
- r is the distance between the particles measured center to center of the particles;
- σ is the distance between the two particles at which the interatomic potential is equal to zero. This parameter indicates how much can the two particles get close;
- r_{max} is the maximum separation at which the particles still have an effect on each other. For greater distances, the interaction is assumed to be zero. Realistically, there

is always some interaction between two particles in space, but it is so small that this approximation is acceptable.

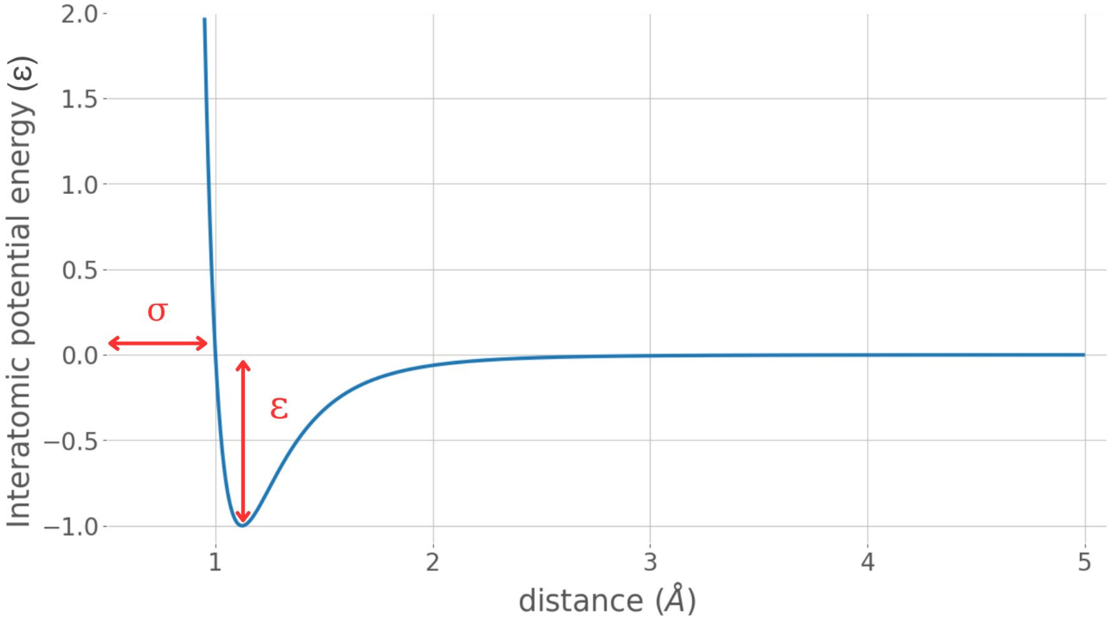


Figure 3.2: The Lennard-Jones potential. The well depth ϵ and the distance parameter σ are represented. The units used for the distance and the potential energy are respectively the Angstrom ($1\ \text{Å} = 1,0 \cdot 10^{-10}$ m) and the pair well depth (ϵ).

As it can be seen in Equation (3.3), the Lennard-Jones model consists of two parts: a steep repulsive term $\left(\frac{\sigma}{r}\right)^{12}$ that dominates in short range distances and a smoother attractive term $\left(\frac{\sigma}{r}\right)^{6}$ that emerges as the distance increases. Figure 3.2 illustrates the behaviour of the function. When the particles are at an infinite distance apart, their interaction is so minimal that the potential energy is considered to be zero. As the distance decreases, the strength of the interaction becomes bigger and the particles come closer until they reach an equilibrium where the minimal potential energy is reached and the resulting force acting on the particles is zero. At this point, the pair of atoms is at its most stable configuration and it will remain in that position unless an external force is exerted on them. If the two atoms are further pushed towards each other and their relative distance becomes smaller that the optimal separation, the electrons orbitals of the two atoms start to interfere with each other; this generates a repulsive force which grows greatly as the separation between the atoms gets smaller.

3.1.2. Oscillating Pair Potential

The problem with the LJ potential is its relatively limited applicability to real, complex systems. In this research, where the interest is mainly focused on how IQCs structures take shape, accuracy is the first priority, and therefore it is important to use a potential formula that accurately reproduces the interatomic interactions between particles in materials where IQCs often occur. Mihalkovič and Henley proposed for this purpose the family of oscillating pair potentials (OPP), a "six parameters analytical form which gives a remarkably faithful account of many intermetallic compounds" [19]. These potentials have the form:

$$V(r) = C_1 r^{-\eta_1} + C_2 r^{-\eta_2} \cos(kr - \phi) \tag{3.4}$$

The first term describes the short-range repulsion; the second captures the medium-range behaviour (the first well's position and depth) and the long-range damped oscillatory behaviour. The parameters η_1 and η_2 are the relative weights of the two terms, while k and ϕ represent the frequency of the oscillation and the phase shift respectively.

Among the potentials in this family, the 3 wells oscillating pair potential (3w-OPP) has been successfully adopted to robustly reproduce the self-assembly of IQCs by Engel et al. [6]. This is expressed as follows:

$$V(r) = \begin{cases} \frac{1}{r^{15}} + \frac{1}{r^3}\cos(k(r-1.25)-\phi) - V(r_{\text{cutoff}}), & \text{if } r \leq r_{\text{cutoff}} \\ 0 & \text{if } r > r_{\text{cutoff}} \end{cases} \tag{3.5}$$

This formula is obtained from the original family of equations 3.4 by setting the parameters $C_1 = C_2 = 1$, $\eta_1 = 15$ and $\eta_2 = 3$, and by introducing two variations: the potential is first truncated at the third maximum occurring in the function such that the potential consisted of three wells, and then axially shifted to zero.

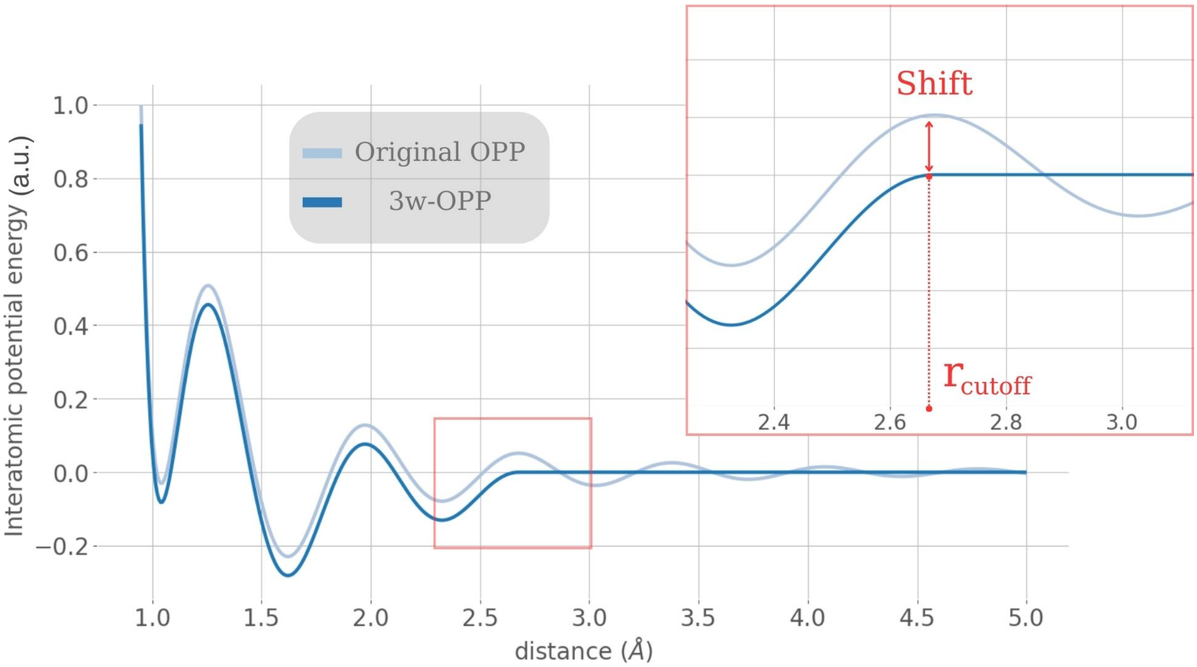


Figure 3.3: Comparison of the original OPP family and the 3w-OPP of [6] using $(k,\phi) = (9,0.4)$. At the value for r when the third maximum of the function is reached, the potential is truncated and it is latitudinally shifted so that the value of V(r) settles on zero for $r \geq r_{\text{cutoff}}$.

The choice to truncate the potential after three wells was made after investigating the robustness of the self-assembly of IQCs variating the potential range r_{cutoff}. The range of the potential was set at the n^{th} maximum for different values of n. Performing different simulations for various values of k and ϕ, the probabilities for forming an IQC were considerably higher for $n = 3$ than for the other values of n tested. Because of the truncation of the potential, for $r \geq r_{\text{cutoff}}$, $V(r) = 0$ and $F(r) = 0$. Then the second operation is required mathematically: in order for the force to be continuous at r_{cutoff}, for $r \leq r_{\text{cutoff}}$ $V(r)$ has to be defined so that $\lim_{x \to r^-_{\text{cutoff}}} V(r) = \lim_{x \to r^+_{\text{cutoff}}} V(r) = 0$ and $\lim_{x \to r^-_{\text{cutoff}}} \frac{\partial V(r)}{\partial r} = \lim_{x \to r^+_{\text{cutoff}}} \frac{\partial V(r)}{\partial r} = 0$.

The parameters k and ϕ also influence the type of structure that forms. In [6], a parameter space for k = [0.38, 0.8] and ϕ = [5.5, 9.5] is considered, and the range for which IQC structures assemble is investigated. Apart for intermediate values of k and high values of ϕ, in which only disordered configurations are found, the rest of the parameter space yields particles to robustly and repeatedly self-assembled in either crystals or quasicrystals. An illustration is given in Figure 3.4.

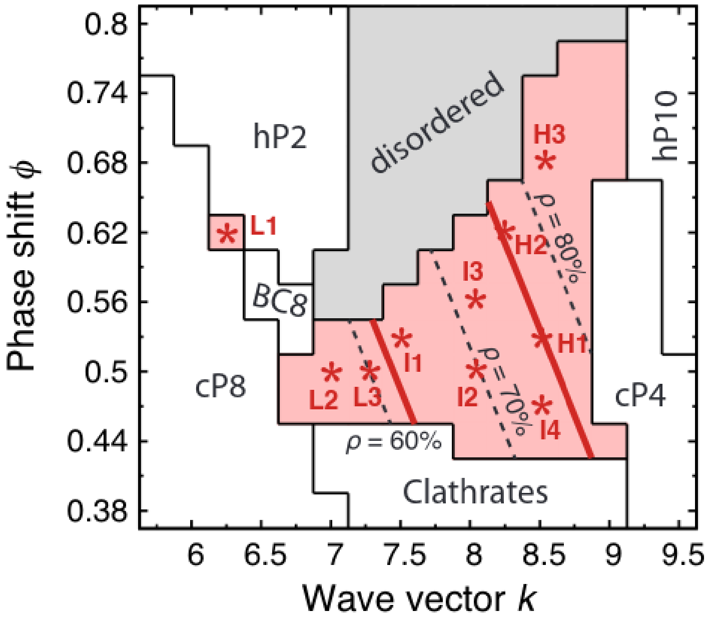


Figure 3.4: Assembly map of the 3w-OPP. The red region highlights the parameter space where icosahedral quasicrystals occur. Taken from [6].

The IQCs region in the assembly map is subdivided in three parts depending on the density of the structure: low (LD), intermediate (ID), and high (HD) density. The LD area is suggested to be the one where the IQCs formed are the most uniform and stable.

However, because of some problems with the optimization algorithm, it was opted to use values of k and ϕ from the ID region, where the quasicrystal structures are less uniform, but still stable. In particular, the parameters used in the simulations to obtain the results presented in this report are $(k, \phi) = (8, 0.5)$.

3.2. Molecular Dynamics

If the potential formula represents the rule that dictates how atoms attract and repel, molecular dynamics is the representation of the movement itself. First performed in the 1950s [20], The idea behind Molecular Dynamics (MD) is that we can study the time evolution of a N-particles system numerically by solving Newton's equation of motion for all the particles at every time interval.

The motion of the atoms is computed by summing the forces exerted on all the atoms at a certain time step due to the interaction potential between the particles. Integrating the resultant force acting on each atom, the new positions and velocities are derived. Then the procedure is repeated for the next time step, until the final time is reached. We look more into details of what Molecular Dynamics comprises, based on the work of Frenkel and Smith in *Understanding Molecular Simulations* [21]. Consider a system of N atoms. The procedure consists of four main steps:

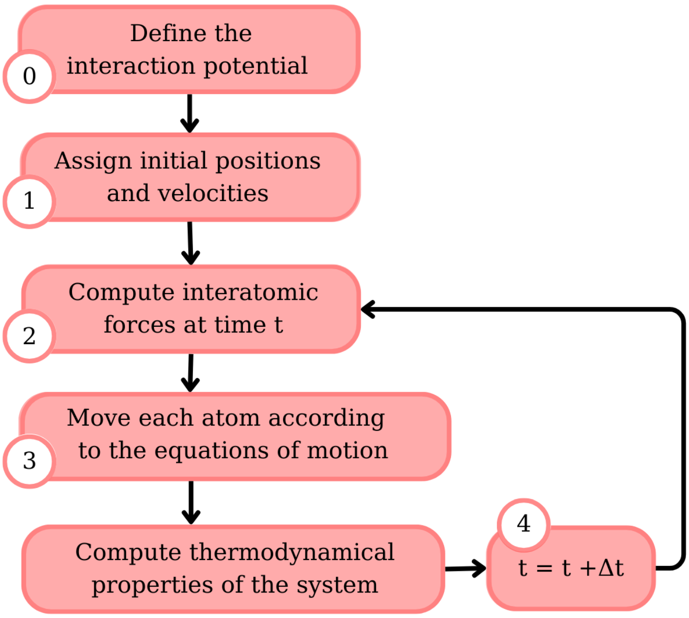

Define the interaction potential
0
Assign initial positions and velocities
1
Compute interatomic forces at time t
2
Move each atom according to the equations of motion
3
Compute thermodynamical properties of the system
4
t = t +Δt

Figure 3.5: Flow chart of Molecular Dynamics

1. **Initialization.** The algorithm starts by assigning all the atomistic properties to each atom: position, mass, velocity, radius. In particular, the initial positions of the atoms are chosen so that they don't overlap, in order to avoid numerical problems due to the extremely big values of the potential between particles at very short distances.

2. **Forces calculation.** This represents the most time-consuming part in the algorithm. The current x, y, z distances between each pair on particles are computed, and then used in the computation of the interactions using an interaction potential formula that depends on the material being modeled. From the potential, the force is derived; this has been thoroughly discussed in Section 3.1. The net force exerted on an atom is then the vector sum of the individual forces arising from the interaction of that atom with any other in the system. Performing this for all the N particles in the system, the number of computations required becomes proportional to $\frac{N(N+1)}{2} = \mathcal{O}(N^2)$.

3. **Configuration update.** The net force F_i acting on the i^{th} atom (for $i = 1, ..., N$) is known; now the positions and the velocities of all the atoms after a certain time step Δt can be derived. This is done by integrating the Newton's laws of motion. Recall that Newton's second law states that $F = m \cdot a$. The acceleration a is the second derivative of the position r with respect to the time t. Then Newton's law for each

particle can be written as:

$$F_i = m_i \frac{\partial^2 r_i}{\partial t^2} \tag{3.6}$$

Rearranging Equation (3.6),

$$\frac{\partial^2 r_i}{\partial t^2} = \frac{F_i(r)}{m_i} \tag{3.7}$$

This system of equations cannot be solved analytically, therefore the solution has to be obtained numerically using an integrator. This is a numerical algorithm that, given the differential equation 3.7 and the approximate position r and velocity v at time t, calculates a good approximation for the integral of $F(r)$ at $t + \Delta t$ assuming that over the small increment Δt, F is constant. There are many algorithms designed for this purpose; the one used in the simulations in this research is the so-called Verlet algorithm [22], a simple, explicit, 2nd order accurate method. The algorithm is derived from Taylor expansions of the coordinate of a particle around time t:

$$\begin{aligned} r(t+\Delta t) &= r(t) + r'(t)\Delta t + r''(t)\frac{\Delta t^2}{2} + r'''(t)\frac{\Delta t^3}{3!} + \mathcal{O}(\Delta t)^4 \\ &= r(t) + v(t)\Delta t + \frac{F}{2m}\Delta t^2 + r'''(t)\frac{\Delta t^3}{3!} + \mathcal{O}(\Delta t)^4 \end{aligned}$$

and

$$\begin{aligned} r(t-\Delta t) &= r(t) - r'(t)\Delta t + r''(t)\frac{\Delta t^2}{2} - r'''(t)\frac{\Delta t^3}{3!} + \mathcal{O}(\Delta t)^4 \\ &= r(t) - v(t)\Delta t + \frac{F}{2m}\Delta t^2 - r'''(t)\frac{\Delta t^3}{3!} + \mathcal{O}(\Delta t)^4 \end{aligned} \tag{3.8}$$

Summing the two equations in 3.8, it yields:

$$r(t+\Delta t) + r(t-\Delta t) = 2r(t) + \frac{F}{m}\Delta t^2 + \mathcal{O}(\Delta t)^4$$

that is equivalent to:

$$r(t+\Delta t) \approx -r(t-\Delta t)2r(t) + \frac{F}{m}\Delta t^2. \tag{3.9}$$

Note that the velocity is not employed in the computation of the new position. However, the velocity can be derived by subtracting the equations in 3.8 which yield:

$$r(t+\Delta t) - r(t-\Delta t) = 2v(t)\Delta t + \mathcal{O}(\Delta t^3)$$

or

$$v(t) = \frac{r(t+\Delta t) - r(t-\Delta t)}{2\Delta t} + \mathcal{O}(\Delta t^2). \tag{3.10}$$

It is important to mention that for the approximation to be accurate, Δt has to be sufficiently small for the method to be numerically stable; this is not granted for any value of Δt, as the Verlet algorithm is an explicit method. Typically, the time steps used are between $1 \cdot 10^{-12} s$ and $1 \cdot 10^{-15} s$ [23]. In this research, the time step used is equal to $\Delta t = 0.5 \cdot 10^{-14}$.

4. **Repeat.** The algorithm proceeds to the next time step and repeats the previous steps listed. The loop will interrupt once the final time step is reached.

One more aspect of MD one can vary is the conditions under which one wants to simulate the system. These conditions are referred to as *ensembles* [23]. Normally, a standard MD simulation is set such that the total energy and the total momenta of the system are conserved. Hence, the simulation is performed in a microcanonical ensemble (NVE), where the number of particles (N), the volume (V), and the total energy (E) are constant. This ensemble represents an isolated system which doesn't exchange matter or heat with the external environment. This however doesn't represent the most appropriate or realistic set-up for a simulation. Indeed, in real experiments it usually is very hard to maintain the energy of the system constant. An alternative, that has also been used to perform the simulations presented in this report, is provided by the canonical ensemble (NVT). In this ensemble, the particles (N), volume (V) and temperature (T) are kept constant. In this case, the system is allowed to exchange heat with the outer environment so that the temperature stays constant.

Molecular Dynamics has some advantages over the existing experiment techniques: first it provides access to detailed information of the quantities relative to each atom at a very fine time resolution that could never be measured directly in reality, and it allows to modify them to control the conditions of the system; thanks to this, it can be decided to perturbate the system in different ways with a precision higher than any existing experiment. In this project, it is extremely valuable to be able to perform Molecular Dynamics as it allows to precisely know the position of each atom in the system, and to measure the energies of intermediate configurations obtained during the simulations to find those for which the potential energy is minimal, and that thus are the most stable.

4

Geometry Optimization

Once the interatomic potential and MD simulations are defined, it is possible to investigate what are the most energetically favorable configurations that a given multi-particle system can assume. Section 4.1 will introduce the Potential Energy Surface (PES) and explain how the minima in this function relate to the stable configurations of the cluster. Then in Section 4.2 different local optimization algorithms designed to find a stable equilibrium of the cluster given an initial configuration will be presented. Finally, in Section 4.3 it will be explained how the local optimization algorithm is utilized in the global optimization, an algorithm that aims to find the global minimum of the system.

4.1. PES

As mentioned previously, every atomic cluster has an energy associated to it. The lower the energy of the cluster is, the more favorable. The process used to find these particular structures is called geometry optimization. It is based on changing the system's geometry (the coordinates of the atoms) to minimize the total energy of the system. Given a cluster formed by N atoms, each pair of particles has a potential energy associated to it. Summing the potential energies of all the possible pairs, the total potential energy of the system is obtained. This energy is described by the potential energy surface (PES), a function depending on the positions of the atoms relative to each other. The position of each atom has three degrees of freedom, the coordinates x, y, and z. However, the PES doesn't depend on the absolute position of the atoms but on the position relative to each other, which causes the system to be invariant under translation and rotation, operations that have each 3 degrees of freedom. Hence, the PES has

$$3N - 5 \tag{4.1}$$

degrees of freedom for a linear system. For a system of two atoms, the PES is a 2-dimensional function depending on the distance between the two atoms. Indeed, for such a system the Potential Energy Surface corresponds to the potential energy as a function of the distance between the two particles. Figure 4.1 shows the PES along with the configurations of the cluster corresponding to the minima.

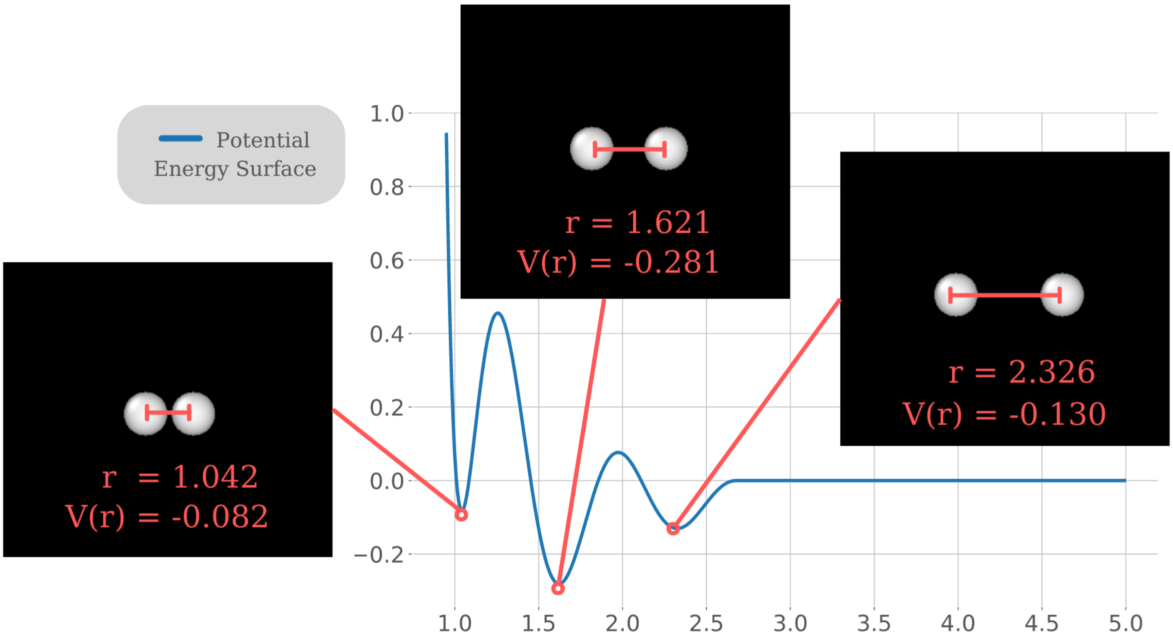


Figure 4.1: Minima corresponding to the minima of the potential energy surface (PES) of the 3w-OPP using $(k, \phi) = (8, 0.5)$ for a cluster with two atoms.

For a cluster with three or more atoms, it is not possible to give a visual representation of the potential surface as it would have to be represented in a 4+ dimensional space.

Investigating the minima of the potential energy surface will yield the structures where the atoms settle on a mechanically stable equilibrium.

4.2. Local Optimization

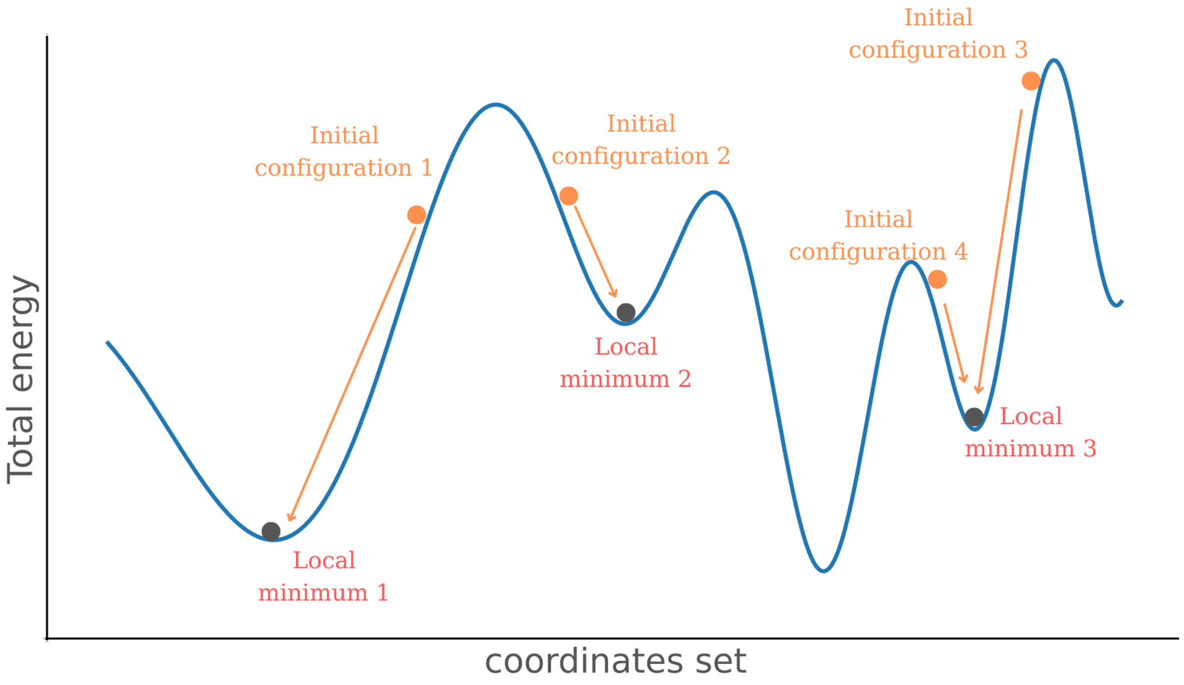


Figure 4.2: Local optimization.

In computational chemistry, geometry optimization has mostly been employed to find the local minima, i.e. observe the energetically stable configuration that a cluster of atoms tends to assume, given an initial configuration (see Figure 4.2). This has for example been applied in [6] where it has been demonstrated that, using the 3w-OPP, a cluster in a random arrangement can settle on a quasicrystalline structure.

There is a number of optimization methods designed to tackle the problem of finding the local minima of a multivariable function. Most of them are based on the determination of some approximation of the Hessian matrix:

$$H_f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(\mathbf{x}) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(\mathbf{x}) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(\mathbf{x}) & \frac{\partial^2 f}{\partial x_2^2}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(\mathbf{x}) & \frac{\partial^2 f}{\partial x_n \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(\mathbf{x}) \end{bmatrix} \tag{4.2}$$

This is then employed to compute a search direction for the minimum. One approach widely used is the Steepest Descent, which iteratively updates the values of the independent variables by moving in the direction opposite to the gradient ∇f, which makes the algorithm gradually converge to the local minimum; another popular method is the Conjugate Gradient (CG), which uses, additionally to the gradient, orthogonal directions to converge faster and thus optimizes efficiency. There are further many other methods like all the Newton methods, et cetera. A more detailed description of the most popular line-search optimization methods can be found in [24].

An alternative is provided by the algorithm proposed by Bitzek et al. [25], a simple and powerful method based on a Molecular Dynamics scheme for structural relaxation, the so-called Fast Inertial Relaxation Engine, or FIRE algorithm. Previously, there already existed optimization methods obtained by modifying MD simulations: some for example minimized the total energy by systematically removing kinetic energy from the system. However, they were not comparable performance-wise with the other methods previously mentioned and were thus not used as primary tools in research. FIRE instead has proven to compete with popular methods for large systems such as CG and L-BFGS [24], and in the case of CG to even overtake it [25]. The method is based on a MD simulation, with no limitations on the integrator used for updating the positions. The trajectories of the particles are continuously updated applying two readjustments on the velocities:

- velocities update:
$$\mathbf{v} = (1-\alpha)\mathbf{v} + \alpha\hat{\mathbf{F}}|\mathbf{v}| \tag{4.3}$$
where $\hat{\mathbf{F}}$ is a unit vector, and
- stop to the motion as soon as $\mathbf{F}(t)\cdot\mathbf{v}(t) < 0$, to prevent uphill motion.

4.3. Global Optimization

Knowing a local minimum is not very useful per se for the prediction of a cluster's structure. The number of minima on the potential energy surface of a cluster of atoms grows exponentially with the number of particles, and therefore a local minimum may not be even close to the most stable configurations that the cluster can assume. Instead, the global minimum represents a much more insightful information, not only because it represents the most stable configuration, but also because its energy can be used to evaluate the importance of the known minima based on their energy difference.

The basic idea behind a global optimization algorithm is to start with an initial configuration of particles and iteratively explore the solution space by making random changes to the configuration. These changes can include modifying some particles' positions, letting the particles move under the influence of a certain temperature, or any other relevant operation. There are several methods that have been designed for such purpose. Many global optimization algorithms are based on thermodynamic principles: one example is the simulated annealing [26], where the parameter space is explored by mimicking the annealing process, where a material is heated and then slowly cooled to reduce defects and reach a more ordered state. Another example is the basin hopping method [27], where the configurations are perturbed by applying random changes to the system's coordinates. These methods come however with some limitations: first, the acceptance condition has an exponential decay for big values of ΔE which make the crossing of high barriers in the PES extremely difficult and thus restricts the search of minima. Another problem is that many configurations are visited repeatedly, and this may lead the algorithm to get stuck between two configurations.

Then there is the genetics algorithm [28], inspired by the natural process of the *survival of the fittest*. This algorithm looks simultaneously for different local minima starting from various initial configurations; then parts of two or more found configurations are merged together to create "mutated" configurations that will be then locally optimized, starting the cycle again.

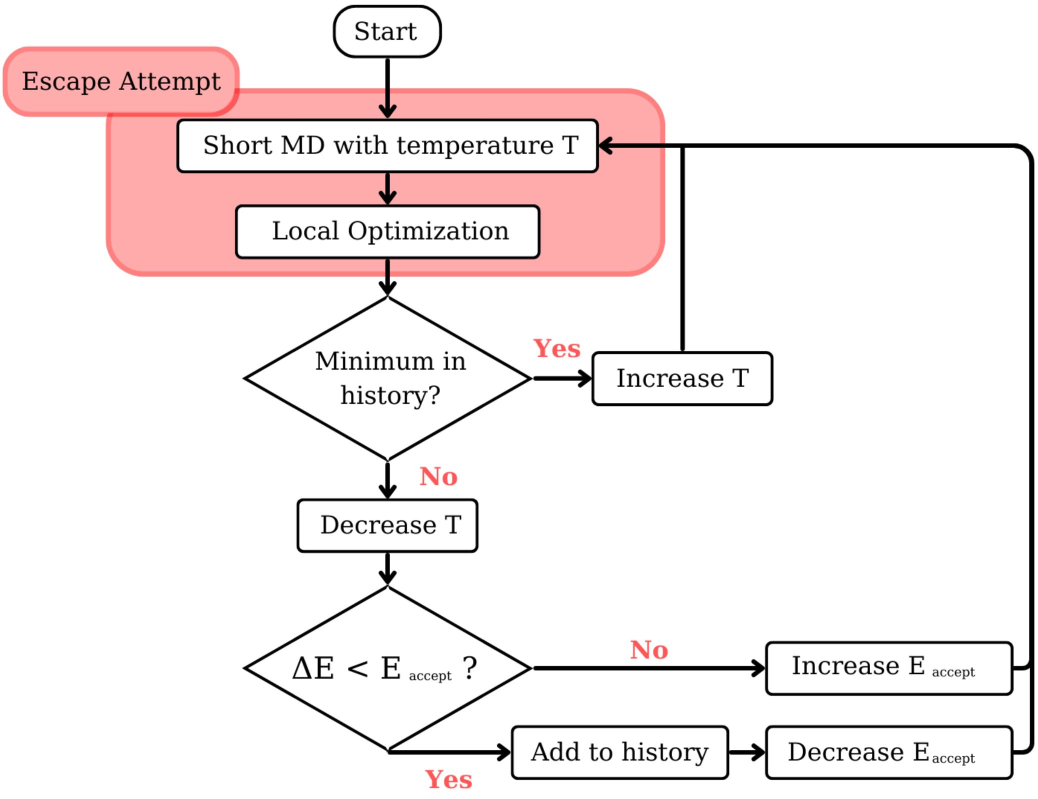


Figure 4.3: Flowchart of the Minima Hopping algorithm. Taken from [29].

For this project, the method chosen is Minima Hopping, first proposed by Stefan Goedecker in 2004 [30]. In this method, the new configurations to be locally optimized are obtained by running a short Molecular Dynamics simulation on the previous minimum found; this offers an advantage over those methods where the new initial configurations are chosen without a physical basis. Moreover, the parameters involved in this method are relatively few, avoiding the problem of optimizing the parameters.

Figure 4.3 offers a schematic view of the steps of the algorithm. The algorithm is formed by two nested loops. The inner loop performs escape attempts until a new configuration not previously stored as minimum is found. The outer loop then either accepts or rejects this potential new minimum depending on the energy difference between the optimized configuration's and the initial configuration's energies. When the new minimum is accepted, the initial configuration is updated equal to the new minimum found. An escape attempt consists on running a short MD simulation (in this case of 250 time steps $\Delta t = 0.005$ fs), then the resulting configuration is optimized locally; the local optimization stops when it converges on a local minimum. To determine whether the minimum found in the current local optimization is new, it is compared to the previously found minima, stored in a database. There are multiple options for the form that the minima are stored in: in fact, equal configurations should be recognized regardless of rotation, translation or permutation of equivalent particles. Therefore, the configurations have to be represented in a form that is invariant under these operations. Such an abstract representation is called a descriptor. The database can then store and compare descriptors instead of the coordinates. For sake of simplicity, in this project it was chosen to use the total potential energy of the system: it can serve as a descriptor, because it can be determined with sufficient accuracy to be considered as unique.

When an escape attempt fails, the temperature is increased by a factor of $\beta = 1.1$, $T_{\text{new}} = \beta T_{\text{old}}$. By gradually increasing the temperature, the MD will eventually make the cluster escape the current well and discover a new minimum. If the escape attempt is successful, the temperature is decreased by a factor of β, i.e. $T_{\text{new}} = \frac{1}{\beta} T_{\text{old}}$, and the energy of the new minimum is considered. A new minimum with a lower energy than the energy of the initial configuration is always accepted and stored in the database; on the other hand, if the new minimum has a higher energy, it is accepted only if the energy increase is below a certain threshold E_{accept}. In either case, the value of E_{accept} is adjusted: if the minimum is accepted, new $E_{\text{accept}} = \frac{1}{\alpha} E_{\text{accept}}$ and if it is rejected, new $E_{\text{accept}} = \alpha E_{\text{accept}}$, where $\alpha = 1.1$.

5

Diffraction Patterns

One of the fundamental tools in crystallography that has a crucial role in the analysis of atomic structures is diffraction patterns. Diffraction occurs when a wave encounters an obstacle or a diffracting object, leading to bending and spreading of the wave. In crystallography, a structure is illuminated with a beam of X-rays; the crystal lattice diffracts the waves that get scattered and interfere with each other, creating a pattern. These pattern are employed to extract information about the crystal lattice illuminated. However, if structures are obtained from simulations and not real-life structures, the data available are usually only projections of the structure, images that require processing and analysis to extract meaningful information. Fourier Transform (FT) algorithms have proven to be a powerful tool in image processing for crystallography.

Fourier proved that any periodic function can be written without loss of information as a weighted sum of infinite sinusoids of different frequencies . The Fourier Transform is the mathematical technique that allows to decompose a function into its frequency components. The Continuous Fourier Transform of a function $f(x)$ is expressed as follows:

$$F(u) = \int_{-\infty}^{\infty} f(x)e^{-2i\pi ux}\,dx \tag{5.1}$$

where x is the space and u is the spatial frequency; note that the Fourier transform is a complex number. The plot then shows the magnitude of the Fourier Transform, that is:

$$A(u) = \sqrt{(\mathrm{Re}(F(u))^2 + (\mathrm{Im}(F(u))^2} \tag{5.2}$$

usually represented on a logarithmic scale, for visualization purposes.

In the context of image processing, the Fourier Transform can be used to analyze the spatial frequencies present in an image. Practically, an image is discrete, therefore the concept of a Discrete Fourier Transform (DFT) has to be introduced. The DFT equation is defined as:

$$F(u_k) = \sum_{n=0}^{N-1} f(x_n) e^{\frac{-2\pi i k n}{N}}, \qquad k = 0, ..., N-1. \tag{5.3}$$

Here $f\{x_n\}$ represents the discrete sequence of N function values, and $F\{u_k\}$ is the transformed sequence. The DFT equation calculates the frequency components of the discrete function and represents them as complex numbers.

Since images are two-dimensional, the Fourier Transform (FT) has to be extended to two dimensions, both the continuous (5.4) and the discrete (5.5) form:

$$F(u, v) = \iint_{-\infty}^{\infty} f(x, y) e^{-2i\pi(ux+vy)} \, dx \, dy \tag{5.4}$$

$$\begin{aligned} F(u_k, v_j) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f(x_n, y_m) e^{\frac{-2\pi i k n}{N}} e^{\frac{-2\pi i j m}{M}}, \\ k = 0, ..., N-1, \\ j = 0, ..., M-1. \end{aligned} \tag{5.5}$$

The idea is still the same, x and y are the spatial coordinates and u and v are the frequencies along x and y respectively. With these expressions, we can find the FT of the discrete image.

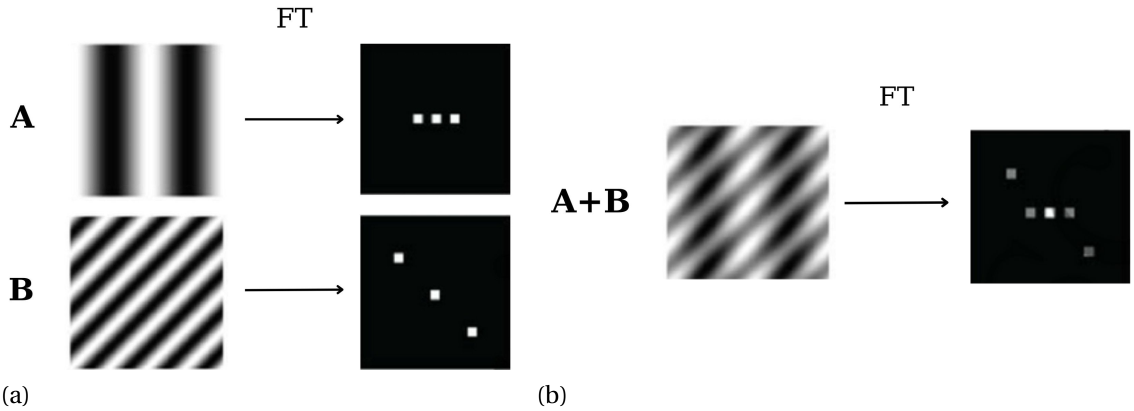


Figure 5.1: (a) shows the Fourier Transform of two sine waves A and B. The corresponding frequency is the distance of the spots in the FT to the center, while the amplitude is given by the brightness of the spots. (b) shows the sum of the two waves A and B, and its Fourier Transform. Taken from [31].

For simple sine waves with frequency k as seen in Figure 5.1a, the two-dimensional FT displays a peak at the center and at the two frequencies $\pm k$. More complex images (see Figure 5.1b) get decomposed in a more complex sum of waves with multiple directions and different amplitudes, and their relative FT becomes less trivial. This can potentially be done for any image.

In the context of crystallography and study of ordered structures in matter, FT are interesting as they exhibit peaks at spatial frequencies of repeated texture. The FT in fact is able to reflect the symmetries and extract the order in a given structure that might not be directly clear to the naked eye.

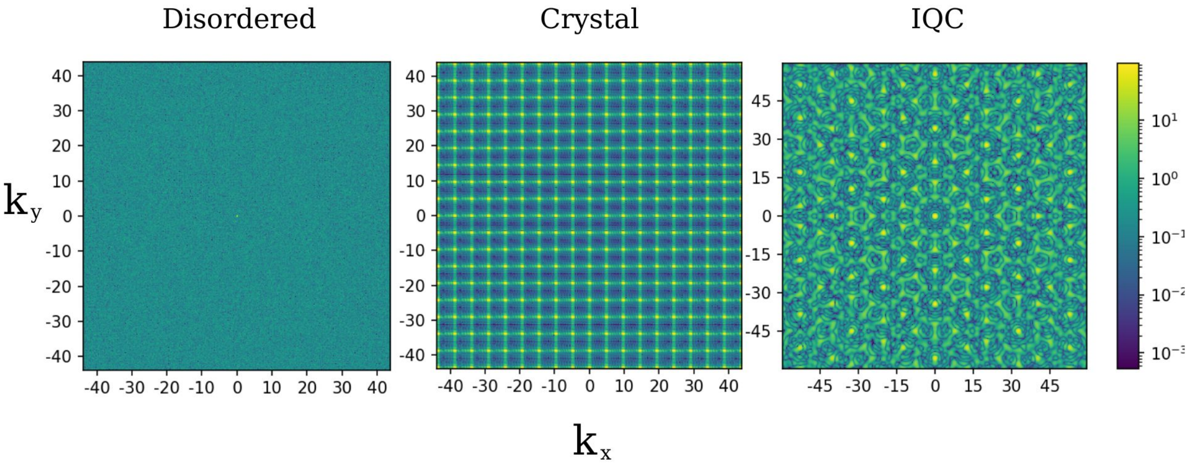


Figure 5.2: Examples of diffraction patterns of clusters with different structures: a disordered cluster, a crystal and an icosahedral quasicrystal.

Figure 5.2 depicts the differences occurring in diffraction patterns depending on the structure considered. The pattern resulting from a disordered structure only exhibits noise; on the other hand, the peaks in the two other patterns reveal that the analyzed structures retain some degree of repeating order. In particular, note that in the diffraction of a crystal structure it's possible to distinguish a 4-fold axis of symmetry, while in the one of the icosahedral quasicrystal the presence of the forbidden 5-fold axis of symmetry is evident.

In this research, the intention was to take diffraction patterns of the obtained structure as a tool to further illustrate and analyze the results. However, it has come up that the clusters considered weren't big enough for the diffraction to highlight some sort of order; therefore, this is kept as a tool that can be employed once this research is expanded.

6

Results

Now that the theory lying behind the tools utilized in this research has been thoroughly discussed, the implementation of such methods can be addressed and the results obtained from the simulations can be shown and discussed. The objective is to run global optimization for clusters with different amount of atoms using both the LJ and the OPP potentials to find for each multiple stable structures that the clusters tend to assume and compare the differences that occur using the two different potentials.

The methods previously discussed have been implemented using the HOOMD-blue Python package. Most of the functionalities needed were already present in this library; an exception was the global optimization algorithm, that had to be built from scratch. The implementations using HOOMD and the global optimization algorithm will be presented in Section 6.1. Then the machinery needed for running the simulations was set up. However, before analyzing the results obtained from the simulations, it was necessary to make sure that the implementations were working as intended. Thus the resulting structures from local optimization simulations using the LJ potential were benchmarked against a database of known LJ global minima structures; this is explained in detail in Section 6.2. Finally, simulations could be run for clusters with few atoms (Section 6.3.1) and then a higher amount of atoms (Section 6.3.2). The latter section will also discuss the memory-managing problems encountered when simulations were run for bigger clusters. At the end, in Section 6.4 it is explored if and how it would be possible to obtain icosahedral quasicrystalline structures starting from a cluster arranged in a specific initial configuration.

6.1. Implementation

First of all, in order to put together all the methods introduced in the previous chapters and translate them into an observable simulation of the arrangement of atoms in a system, a software machinery was needed. For this purpose the package HOOMD-blue was used.

HOOMD-blue is a Python package [1] that performs hard particle Monte Carlo simulations and Molecular Dynamics simulations of particle systems using different types of potentials [32]. Among its features, its main strengths are flexibility, that allows one to control simulation parameters, and its ability to read and write files specifically designed to store the information relative to a cluster, that can easily and interactively be visualized and studied with the support of other software packages.

A simulation is configured by initiating a Simulation object. This object consists of a *state* and of *operations* that are applied on that state during the simulation run.

```
cpu = hoomd.device.CPU()
sim = hoomd.Simulation(device=cpu, seed=1)
```

The *state* contains the particles' properties such as box (i.e. the space where the particles are allowed to move), positions, velocities, et cetera; *operations* is a set of functions aimed to examine or modify the state. Once the simulation has just been initialized, it has no state; this can be imported from a General Simulation Data (GSD) file:

```
sim.create_state_from_gsd(filename)
```

GSD is a file format specifically created for this python package. This file format stores trajectories of the complete system state in a binary file. Moreover, a GSD file can contain multiple frames, and thus a single file can account for the time evolution of the system. The behavior of the simulation stored in a GSD file can be explored using the visualisation tool OVITO (Open Visualization Tool) [33]. An ulterior tool to illustrate the structures is provided by the diffraction patterns, presented in Chapter 5. HOOMD-blue provides a method that can extract the diffraction pattern of a cluster's structure stored in a GSD file by taking the FT of the projection of such structure along the required axis. The code to create and store such plots, given a file path and an orientation, is saved in Appendix A.5.

[1] Note that this project was started using the v3.10.0 version of the package, which has now advanced to the 4.0.1 version; thus in the code there might be some deprecated methods

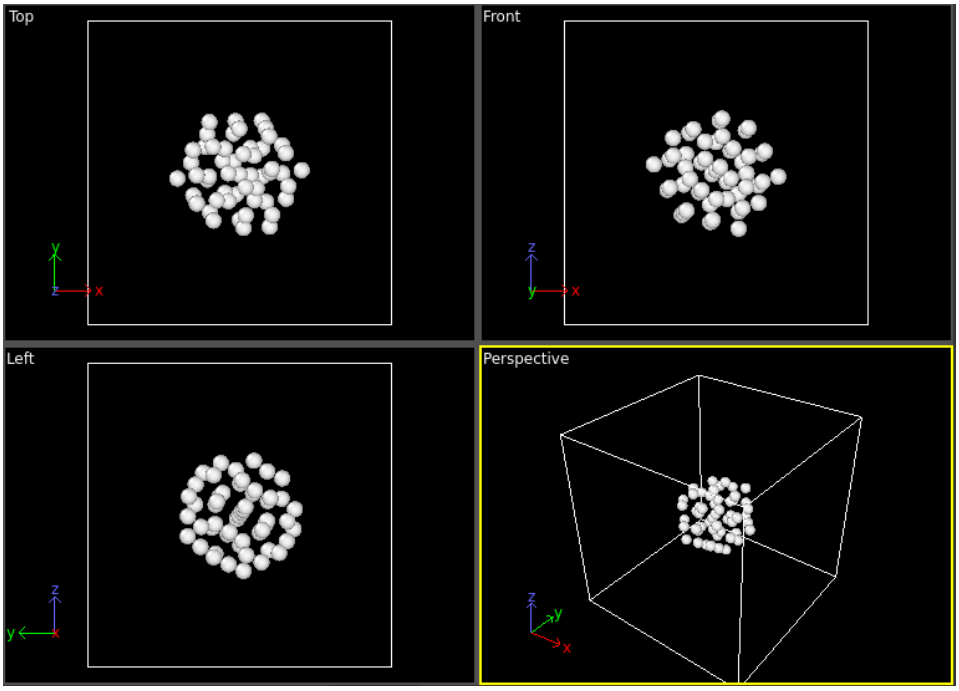


Figure 6.1: Interface of OVITO, when visualizing a cluster of 52 particles.

Molecular Dynamics is implemented in HOOMD-blue by appending to the *operations* of the state an *integrator* object. This integrator must be given a time step as a parameter, and a method (which corresponds to the ensembles mentioned in Section 3.2) and forces (e.g. LJ, OPP) must be appended.

```
integrator = hoomd.md.Integrator(dt=0.005)
integrator.forces.append(pot)
nvt = hoomd.md.methods.NVT(kT=kT, filter=hoomd.filter.All(), tau=tau)
integrator.methods.append(nvt)
sim.operations.integrator = integrator
```

When a simulation of n time steps is instantiated, the integrator computes the net force acting on each particle of the system, and based on that it updates the simulation state on the next time step; as said before, the state includes information regarding the location and velocities of the particles, but also for example the thermodynamic properties of the system.

The forces are derived from one of the several different pair potential energies provided by HOOMD-blue. While pair potentials are nominally defined between all pairs of particles, MD simulations evaluate short ranged pair potentials only for $r < r_{\text{cutoff}}$ to make the computation fast through the use of a neighbor list. Both LJ and OPP potentials are already provided by HOOMD-blue and the relative pair forces can be instantiated by using:

```
opp = hoomd.md.pair.OPP(nlist=cell, default_r_cut=range)
```

```python
lj = hoomd.md.pair.LJ(nlist=cell)
```

However, if one wants to use a less known pair potential, there is the possibility to implement the desired potential and the relative forces manually using the base class of the pair forces [34]:

```python
my_pot = hoomd.md.pair.Pair(nlist, default_r_cut=None, default_r_on=0.0, mode='none')
```

The local optimization is set up in a very similar way to a normal MD simulation; however, instead of assigning to the integrator of the simulation a standard Integrator object, it uses the energy minimizer FIRE, the local optimization algorithm presented in Section 4.2.

```python
fire = hoomd.md.minimize.FIRE(dt=dt, force_tol=1e-2, angmom_tol=1e-1, energy_tol=1e-2, min_steps_conv=1)
fire.forces.append(pot)
fire.methods.append(hoomd.md.methods.NVE(hoomd.filter.All()))
sim.operations.integrator = fire
```

Then the simulation doesn't run for a fixed amount of time steps, but it stops only once the method has converged:

```python
while not(fire.converged):
        sim.run(1)
```

The functions created to run Molecular Dynamics and local optimization can be found in Appendix A.3, along with an explanation of their functioning. These functions are used as building blocks for the global optimization algorithm, that is not pre-implemented in HOOMD-blue. The algorithm chosen is Minima Hopping. The full code can be found in Appendix A.4. The way this algorithm works has thoroughly been explained in Section 4.3. It is important to note that when checking if a minimum is already present in the history, the energies cannot be compared absolutely; that means, we cannot say that two minima are the same if and only if $E_1 = E_2$. This comes from the fact that, given a cluster of a certain size, the value of the energy obtained from the simulation could be slightly different from the one given by the database, due to rounding corrections the computer has to do when computing the energy of the configuration. Therefore instead of using an equality, we consider two minima the same if the difference $\Delta E = |E_1 - E_2|$ is small enough to be neglectable, i.e. $\Delta E < hist\_check$. The parameter $hist\_check$ has a very small value that depends on the size of the cluster: for a bigger cluster, there are more particles that can have small neglectable differences, thus a bigger buffer is needed. The final choice was to set $hist\_check$ equal to 0.5 for clusters of size $N \leq 50$, 1 for $50 < N \leq 500$ and 5 for $N > 500$.

There is one aspect of the algorithm that has not been covered yet: when does the algorithm stop looking for new minima? Ideally, one would want to find all or most of the minima of the given cluster of particles. In practice, this gets increasingly difficult, since the number of minima grows exponentially with number of atoms in the cluster [28]. Moreover, our interest lies not quite in the knowledge of the most stable configuration (i.e. the global minimum), but rather in studying multiple stable configurations that the clusters can assume (i.e. different local minima). After these considerations, it was decided to stop the search of new local minima once a certain quota of different minima was met; this quota depends on the size of the cluster being studied. The algorithm is interrupted for small clusters ($N \leq 500$) after 15 different are found, and for bigger clusters ($N > 500$) after 20. Therefore this algorithm, rather than being used for its original purpose to find the global minimum, is employed to find different stable configurations, so different local minima. It does not guarantee to find the global minimum, but from the structures found it can be asserted with a certain degree of confidence how stable is a certain configuration: for example, if the cluster keeps going back to structures very similar to each other, it can be concluded that these structure are located in a deep well and represent thus a quite stable configuration.

Consequently, it becomes very relevant to the results obtained, the configuration that we start the optimization from. The first batch of simulations, those discussed in Section 6.3, have been run starting from configurations where the atoms are arranged in a sort of single "cloud"; these were taken from the Cambridge Energy Landscape Database [35], presented in Section 6.2. Afterwards, in Section 6.4, the simulations are started from different configurations formed instead by multiple building blocks.

6.2. Benchmark: Cambridge Database

The Cambridge Energy Landscape is a database that contains the results of global optimizations for a variety of systems in a downloadable form. In particular, it contains the coordinates of the global minima obtained using the Basin Hopping algorithm and the corresponding total energy of Lennard-Jones clusters with sizes N = 3-150, 561-1000. The work done on the clusters of size $N < 110$ has been published in [27]. It has been shown in previous studies that the dominant structural pattern in LJ clusters is represented by the Mackay icosahedron [36]. Complete icosahedral structures are possible, for example, for N = 13, 55, 135, 147; for the intermediate sizes, the clusters tend to form an icosahedral core covered by an incomplete layer. This is clearly visualized in Figure 6.2 and Figure 6.3.

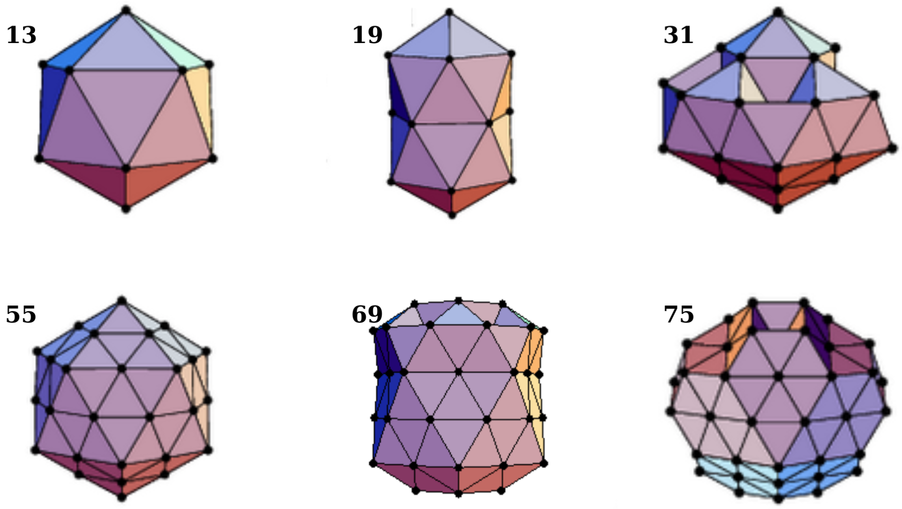


Figure 6.2: Icosahedral global minima for sizes $N = 13$, 19, 31 formed by growth of a layer over the 13-atom Mackay icosahedron and for sizes $N = 55$, 69, 75 formed by growth of a layer over the 55-atom Mackay icosahedron. The images of the structures are taken from [35] and from [37].

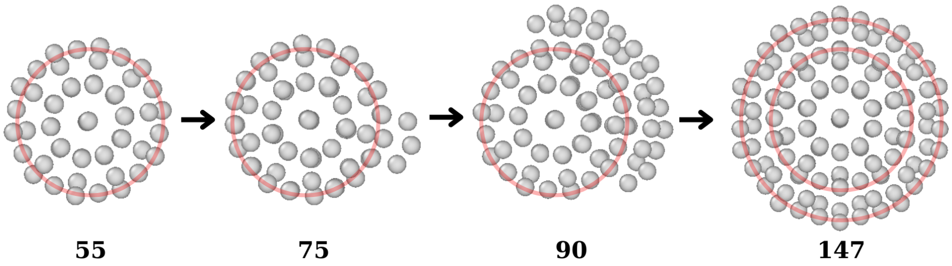


Figure 6.3: Perspective of the global minima of two clusters with a perfect icosahedral shape, 55 and 147, and two intermediate clusters, 75 and 90. In the clusters with 75 and 90 particles it is clear that the "exceeding" particles are forming a layer around the 55 Mackay icosahedron to tend to the 147 Mackay icosahedron.

Note that in Figure 6.3 we can also see the tight correlation between the shape that the LJ clusters get arranged into and the plot of the potential formula: the LJ potential has one and only one minimum in its function, thus only one distance r_{opti} where the particles tend to stabilize on; therefore, it only feels logical that the particles arrange themselves radially around a center in circles that are separated by the same distance r_{opti}.

Being these structures in the Cambridge Landscape Database well-recognized global minima, they can be used to study whether the methods implemented work as they should. for the sake of brevity, these clusters will be referred to as "Cambridge Clusters" from now on. The validation has been performed for a sample of clusters of different sizes out the 839 available. First of all, the Cambridge Clusters have been downloaded and converted into GSD files, using the code in Appendix A.1. After that, for each of the sizes considered for the validation, the relative cambridge file has been opened and the coordinates of one or more random particles have been disturbed, making sure that the disturbed particles do not overlap with other particles and that they are still in the range of action of the potential; the code to perform this is shown in Appendix A.2. Therefore, the structure of the cluster becomes different from the original one, but close enough that it is expected to be mapped back to the Cambridge Cluster's structure once local optimization is performed on it.

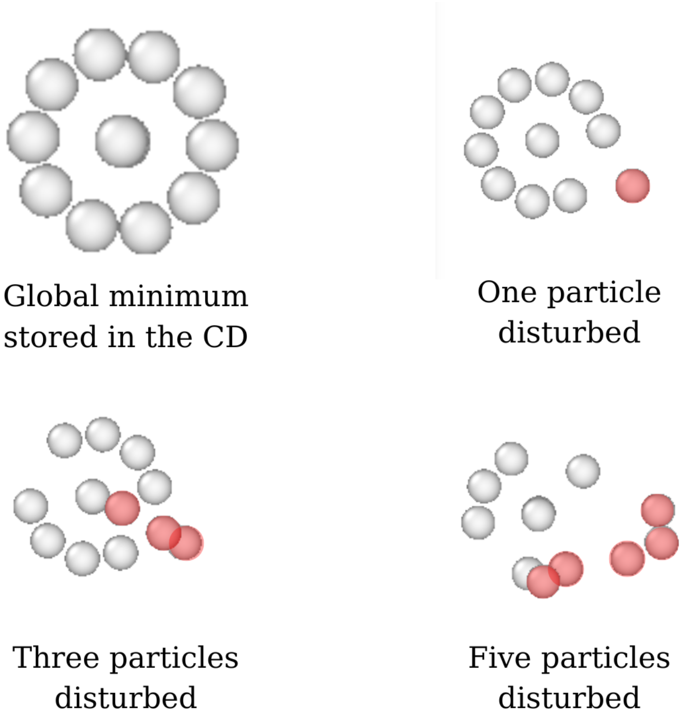


Figure 6.4: Cambridge Cluster of 13 atoms, and the resulting configurations after disturbing the positions of 1, 3, and 5 particles.

The configuration is locally optimized, and the result is compared to the Cambridge Cluster's configuration. A visual comparison already suggests that the found structure is the same as the cambridge one; however, a proper validation is performed by using the energies of the two structures. The two energies are subtracted and the difference ΔE is studied, adopting the same approach used when checking if a structure is already present in the history in the global optimization algorithm: if $\Delta E < hist\_check$, then the two configurations are considered equal. For all the sizes considered, it yielded that the Cambridge Clusters and those obtained by running the simulations were indeed the same.

Potential energy of the minima configurations (ϵ)			
Cluster size	Cambridge Cluster	Global optimization algorithm	ΔE
13	-44.3268	-44.3268	0.0000
26	-108.3156	-107.9741	0.3415
38	-173.9284	-173.8492	0.0792
52	-258.2300	-258.0132	0.2168
75	-397.4923	-396.5559	0.9364
110	-621.7882	-620.8543	0.9339
135	-790.2781	-789.4523	0.8258

Furthermore, running global optimization on these clusters, it always resulted that the other minima configurations found had all a total energy higher than that of the cambridge configuration, as it should be since that represents the global minimum of the system while the newly found configurations are only local minima. These results are enough to conclude that the methods implement provide the expected results.

6.3. Optimization of Cambridge Clusters

6.3.1. Small Clusters

In this section the results obtained running the global optimization using both the Lennard-Jones and the Oscillating Pair Potentials on the Cambridge Clusters of sizes $N \leq 500$ are presented.

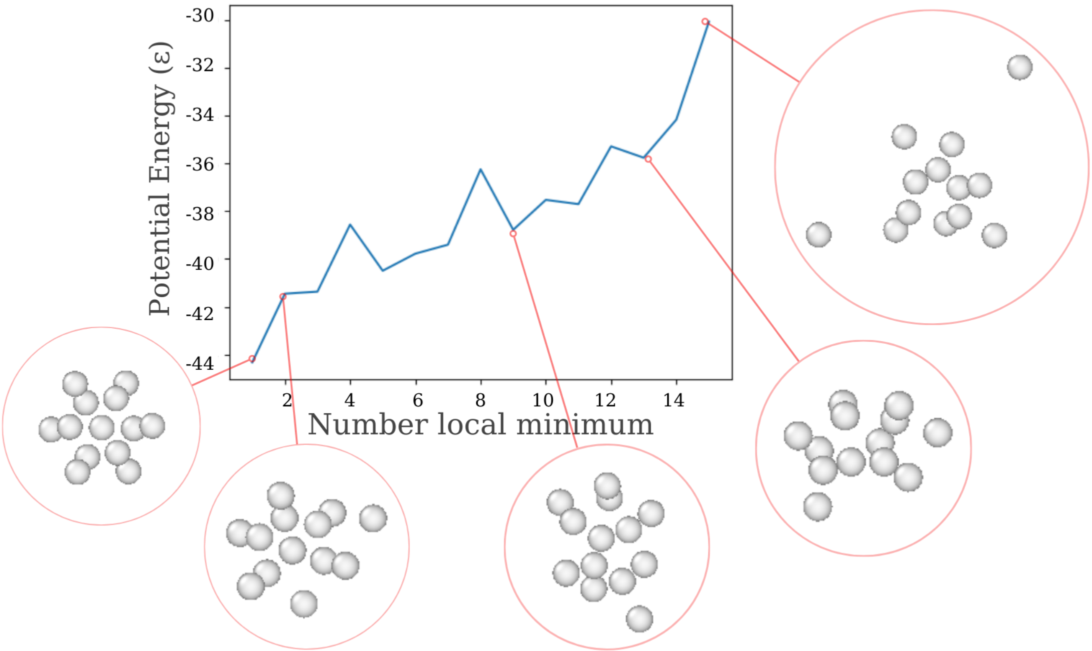


Figure 6.5: Structures of some minima found when globally optimizing the 13 atoms Cambridge Cluster.

For the simulations run using the Lennard-Jones potential, as it was expected, the lowest-energy minimum found corresponded to the configuration stored in the Cambridge database, and the rest of the minima had a total energy higher than that given in the Cambridge Landscape Database. The total energy of the configurations corresponding to the minima found exhibited an increasing trend as the global optimization went on. Figure 6.5 shows the plot of energies of the 15 minima of the cluster with 13 atoms found during global optimization, along with some structures.

Initially, the short Molecular Dynamics simulations disturbed the structure such that most of the particles went back to their position in the global minimum configuration, while few of them instead created some bonds with neighboring particles and stabilized on the shell of the global minimum configuration. When the global optimization algorithm kept seeking more configurations, the local minima found became progressively more disordered, see Figure 6.6, and some particles started to have no interactions with the rest; these structure represent the least energetically-favourable stable configurations.

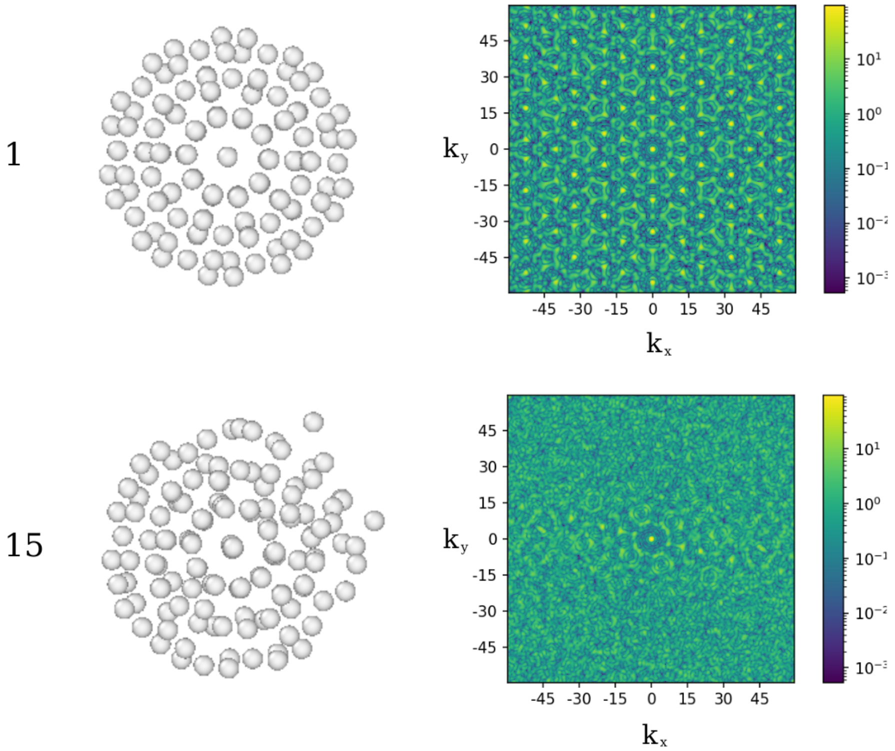
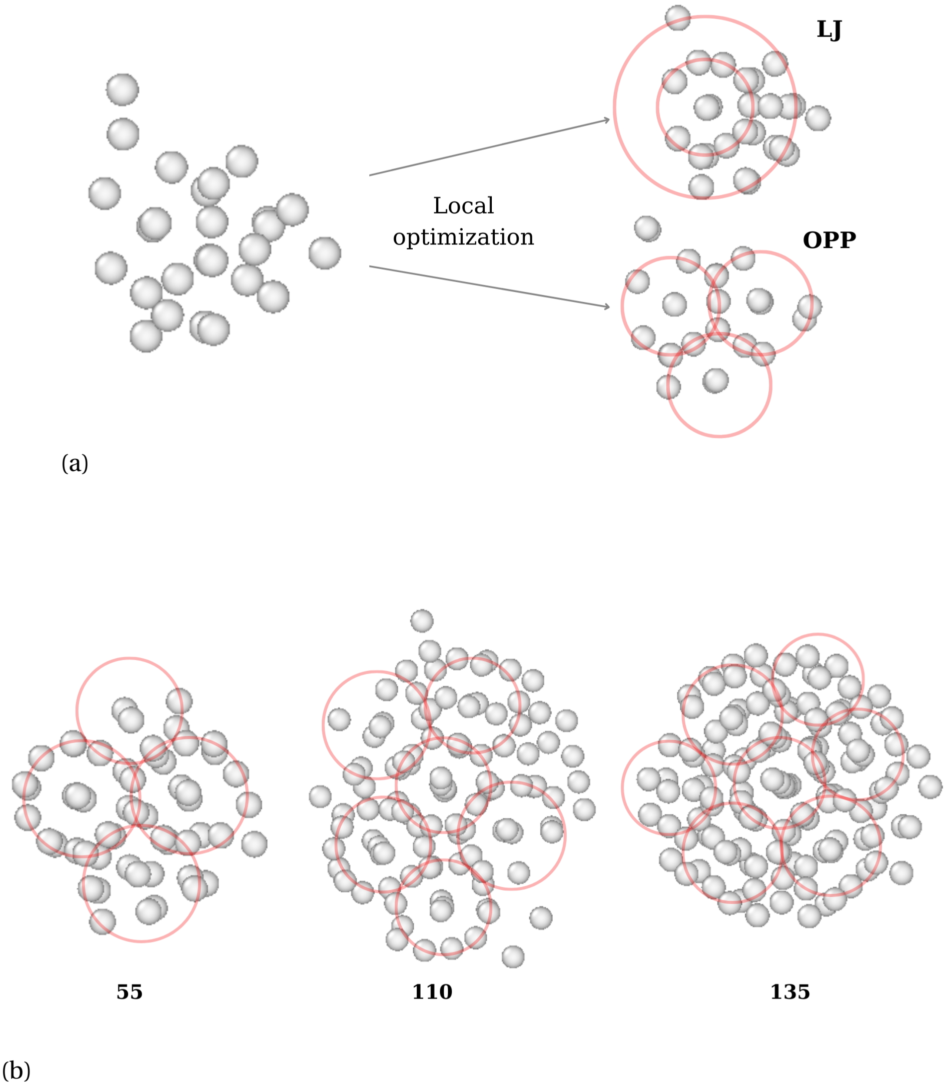


Figure 6.6: Structure and diffraction patterns of the 1st found and last found minima of a cluster of 135 particles. For sufficiently big clusters, the symmetries in the structure can be seen from the diffraction patterns. In the fist minimum (which is the global minimum) it is clearly distinguishable a 10-fold symmetry, typical of icosahedral order. The diffraction of the last minimum on the other hand is much more confused.

Using the Oscillating Pair Potential, already from small clusters it is clear that the obtained results are quite different. The cluster is not very stable on the configuration stored in the Cambridge Landscape Database. Initially it swells, imputable to the fact that, in the PES of the OPP, the Cambridge Cluster's configuration is located in the well that corresponds to a slightly bigger distance ($r = 1.621$) than the $r_{\text{opti}} = \sqrt[6]{2} = 1.122$ of the Lennard-Jones potential; after that, in the minima configurations found, the particles don't tend to form an unique cloud as LJ clusters did, but instead arrange themselves into multiple circular configurations(see Figure 6.7b). This tendency can be explained considering the formula of the oscillating pair potential: as seen in Section 3.1.2, the OPP has three wells, i.e. three optimal distances where the potential energy exhibits a minimum. Therefore, when moved around by the Molecular Dynamics simulations, it is more energy-favourable for the particles not to strive to go back to the global minima configuration where there is one predominant distance between the particles that is r_{opti}, instead there are more intermediate stable configurations where different distances appear. Consider for example a cluster of 26 particles, shown in Figure 6.7a.



(a)

(b)

Figure 6.7: (a) shows a cluster of 26 particles and the locally optimized structures using LJ and OPP. (b) shows clusters of different sizes optimized using OPP, highlighting the tendency of the particles to arrange in multiple circles.

The structure on the left comes from an intermediate step of the global optimization performed using OPP, it is the configuration resulting from a short MD simulation performed on a minimum structure; the two images on the right depict the structures it settles onto when locally optimized using LJ and using OPP. In the LJ structure, it can be clearly recognized a core that is being covered by an overlayer, while in the structure found using OPP, the particle tend to arrange themselves not in a unique cloud but in more circular structures attached to each other. This kind of arrangement becomes more evident as the size of the cluster increases, as it can be seen in Figure 6.7b.

6.3.2. Big Clusters

When it was tried to perform the same global optimization for bigger clusters ($N > 500$), the program encountered memory-managing issues. In fact, the bigger the cluster's size, the more memory needed to store all the trajectories of the MD simulations and the minimization processes. For clusters of size $N > 500$, this caused the process to be killed as the memory request exceeded 8 Gigabytes (that is the size of the RAM in the local machine) before the number of requested minima was reached. This is partially caused by the use of a high-level language like Python, that has a lot of overhead time, but also by the fact that the routines implemented require a lot of memory. The problem was partially solved by moving the simulations on the HPC DelftBlue [38]; in this way the resources that could be allocated for the simulations were significantly increased. It was created a submission script (which can be found in Appendix A.6) that called the global optimization routine for a given cluster size and potential, and allocated for the job 16 Gigabytes of memory:

```
#SBATCH --cpus-per-task=4 # number of cores
#SBATCH --mem-per-cpu=4G  # number of GB per core
```

Using DelftBlue, the problem of running out of memory was solved and the global optimization could successfully conclude to find 20 minima for all the sizes tested, that went up to 1000 particles. Furthermore, it was investigated how advantageous could be the use of the HPC over the local machine. Figure 6.8 shows the wall-clock time for running global optimization with OPP for different cluster sizes on the local machine and on the HPC using a varying amount of cores.

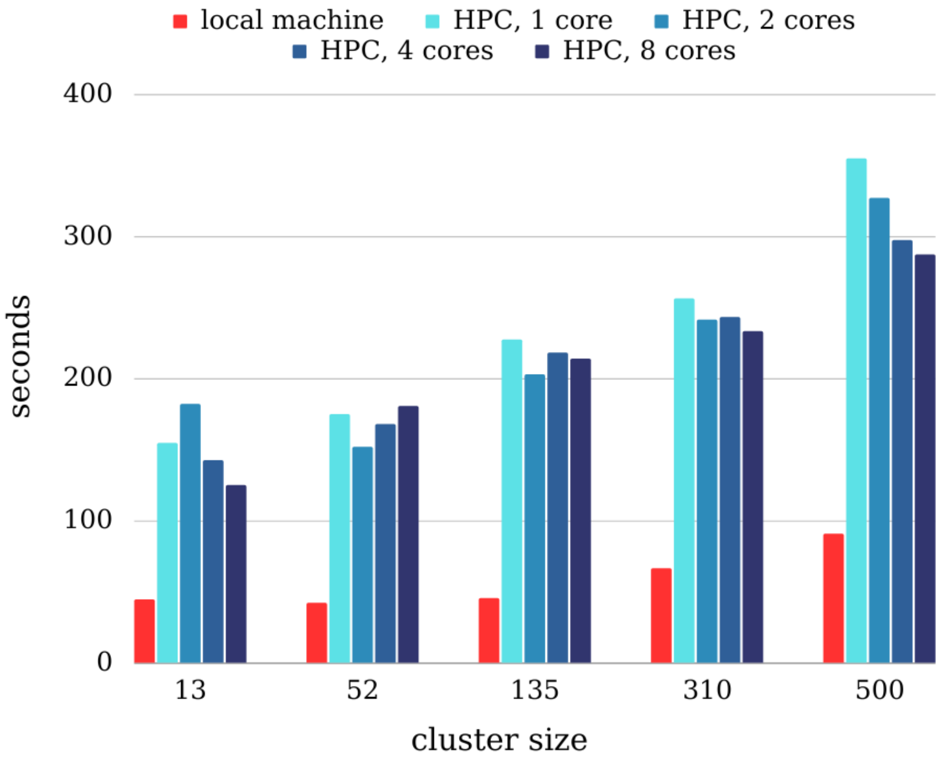


Figure 6.8: Histogram of the wall-clock time taken for the global optimization runs on clusters of sizes $N = 13$, 52, 135, 310, 500.

For small sizes, the use of the HPC is not beneficial; the wall-clock time of the runs on the local machine has an average of 58 seconds, while on DelftBlue the average is more than doubled, at least 202 seconds (average using 8 cores). For clusters over 500 particles, it is not possible to make a comparison as the RAM requirements are too large for a local machine. It is speculated that by the time that the parallelization possible in DelftBlue would actually have an impact on the wall-clock time, the RAM of the local machine goes out of memory, thus that is the reason why it is not possible to see any improvement. Moreover, note that there isn't a significant improvement in the wall-clock time when the number of cores used is increased, i.e. it doesn't matter how many cores are allocated; this is one more proof that this is not a computation-bound program, but primarily a memory-bound program. To conclude, the routines implemented are found to be quite memory-inefficient; if one wants to conduct further studies using this code, this has to be fixed.

Analyzing the results obtained for the big clusters, it was noticed the same tendency described in Section 6.3.1, in which the LJ minima stay arranged in one big cloud, and the OPP minima tend to form multiple circles; as the cluster gets bigger, though, the circular structures do not involve the full configuration but are spread, see Figure 6.9.

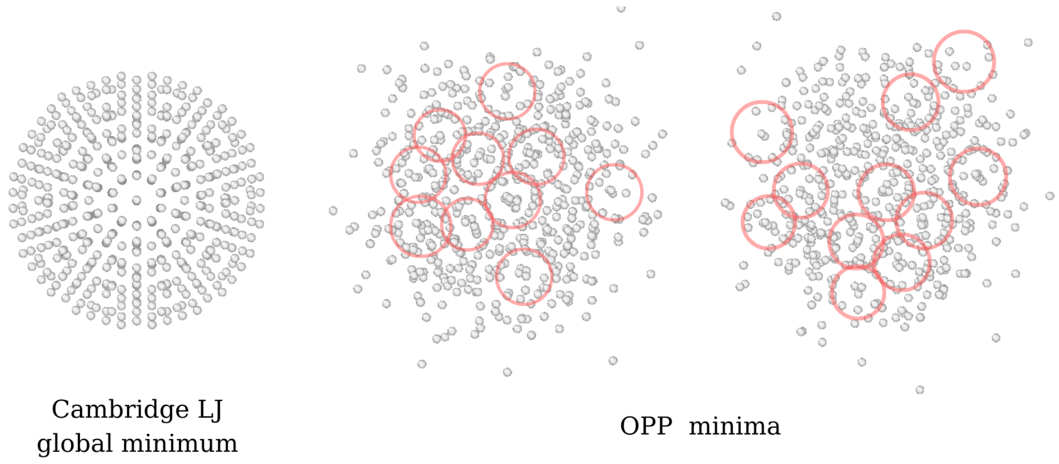
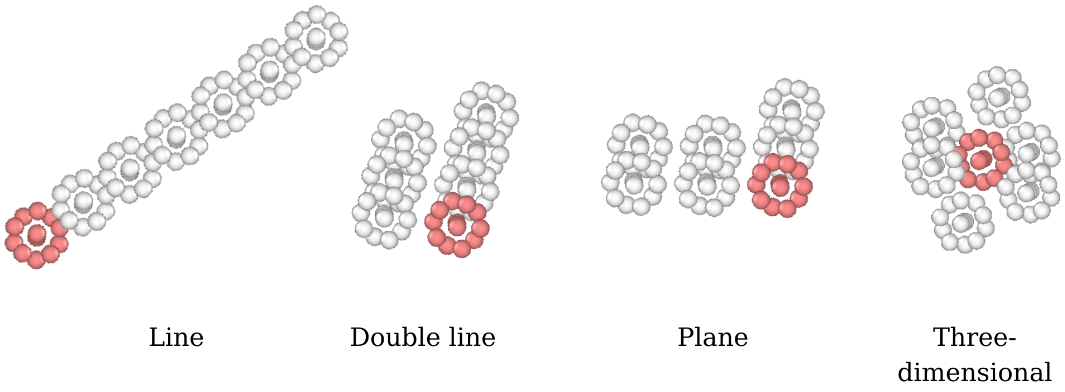


Figure 6.9: LJ global minimum and two OPP minima found during global optimization of a cluster of 550 particles, performed on DelftBlue.

However, for clusters of size $N \geq 650$, another problem arose: the configurations of minima found didn't deviate substantially from the Cambridge Clusters. This meant that the portion of the Potential Energy Surface investigated was but a really limited fraction of the total surface. Three possibles fixes have been identified: first of all, in order to have a more complete representation of the PES, the minima requested in the global optimization should be many more. Moreover, in order to have significantly different structures, the threshold energy difference $hist\_check$ to consider two minima configurations different should have been set higher. Finally, the particles can be allowed to deviate more from the initial configuration by increasing the temperature used in the MD simulations.

6.4. Growth of IQCs Structures

When running global optimization on the Cambridge Clusters, it has been seen that the two potential formula considered deliver different stable configurations: the clusters optimized with LJ tend to form a single cloud while the OPP makes the particles arrange themselves into multiple circular structures. However, there haven't been found structure that clearly represent an icosahedral quasicrystal; there could be multiple reasons for that: for small clusters, there aren't enough particles to actually reproduce some long range order, while for bigger clusters the global optimization algorithm implemented doesn't explore a wide enough portion of the Potential Energy Surface of the cluster. Therefore alternative testing options had to be explored. As mentioned in Section 6.1, it is very relevant for the results obtained the structure that we start the global optimization from. Therefore it was investigated what would happen if, instead of taking as starting configuration the Cambridge Clusters, in which the atoms form a single cloud, we construct ourselves the initial configuration by arranging the system into multiple blocks with icosahedral symmetry put next to each other. One would expect that employing the Lennard-Jones potential will still make the particle tend to coalesce back together into one single cloud; however, it will be interesting to see if using the Oscillating Pair Potential would instead make the particles keep the icosahedral symmetry.



Line

Double line

Plane

Three-dimensional

Figure 6.10: Types of multiple blocks configurations used as starting point for the global optimization, composed by seven blocks of the Cambridge Cluster of 13 atoms. The highlighted particles represent the starting unit block.

First of all, these configurations had to be constructed; for this, a number of different arrangements of the building blocks has been explored. It was created a function (which can be found in Appendix A.7) that would create and store in a GSD file a new configuration, given the size of the Cambridge Cluster taken as building block $N\_original$, the number of blocks desired $clusters$, and a certain type of arrangement $type\_confi$; the possibilities considered for the type of arrangements were to create a line, a double line, a plane, or a three-dimensional configuration. These different arrangements can be visualized in Figure 6.10. For the simulations, it was decided to use as building block the Cambridge Cluster of 13 atoms, which is known to have a perfect icosahedral shape, as explained in Section 6.2. Then it was created a submission script, shown in Appendix A.8 that called a global optimization routine, similar to the one presented before but adapted to first initialize the multiple clusters' configurations; this routine executed the optimization given the desired initial configuration and the potential wanted. The results obtained met our expectations. Figure 6.11 displays an example of LJ and OPP minima and the corresponding starting configuration. The differences will be discussed below. Note that the results shown here will be only those yielded by a cluster of 91 atoms, formed by seven blocks of 13 atoms each; other clusters with a different size (i.e. built with a different amount of blocks of 13 atoms) had very similar behavior.

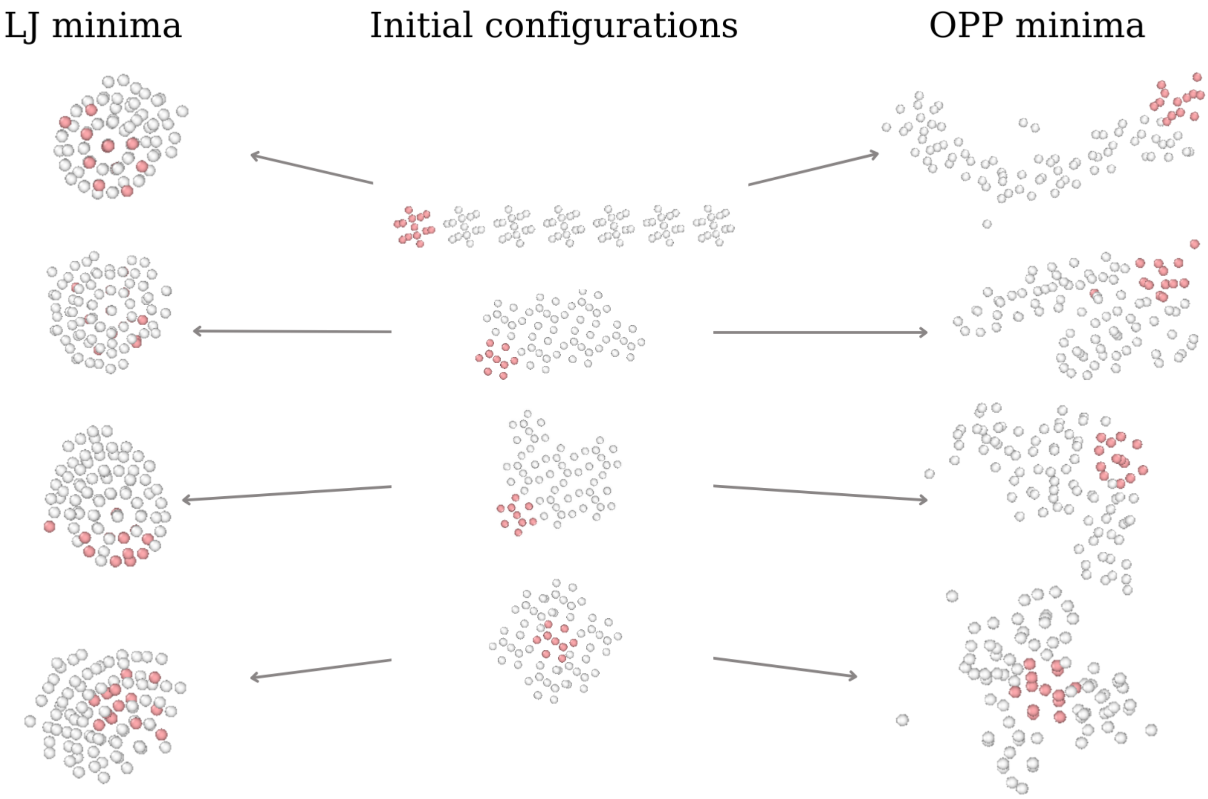


Figure 6.11: Structures of selected minima when optimizing different configuration of 7 blocks of 13 atoms (in total 91 atoms) using LJ versus using OPP.

Using the Lennard-Jones potential, the clusters consistently tended to form a single cloud, independently from the configuration the global optimization started from; this held even when the blocks were arranged in a line, as it can be seen in Figure 6.12.

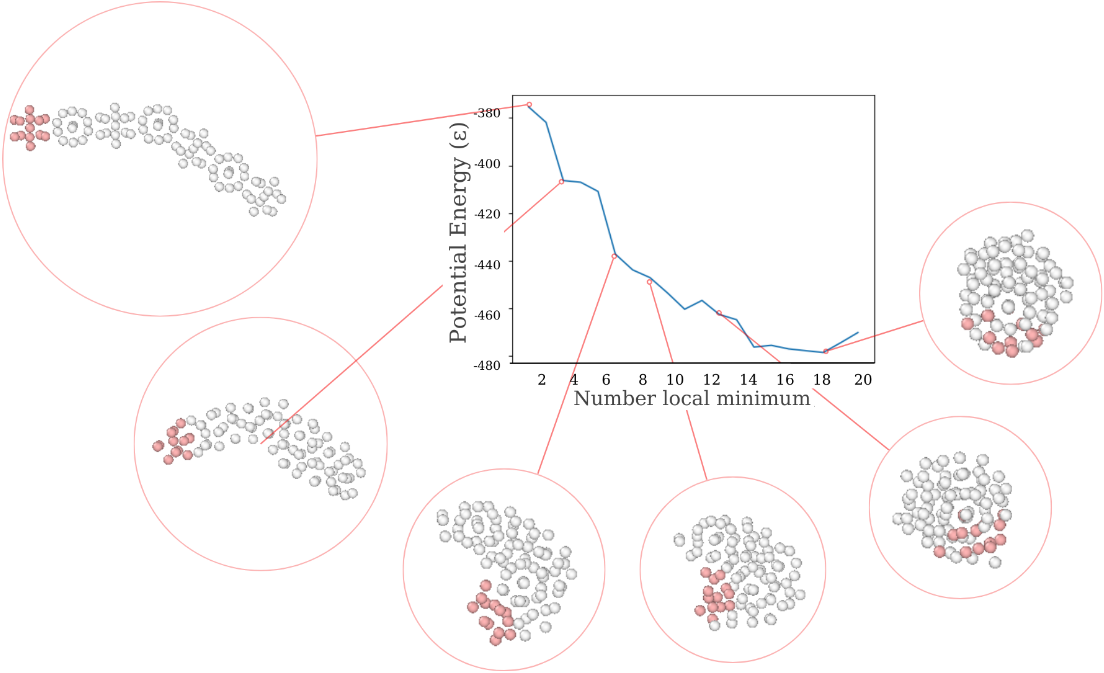


Figure 6.12: Structures of selected minima when the line configuration of 7 blocks of 13 atoms (in total 91 atoms) was globally optimized using LJ.

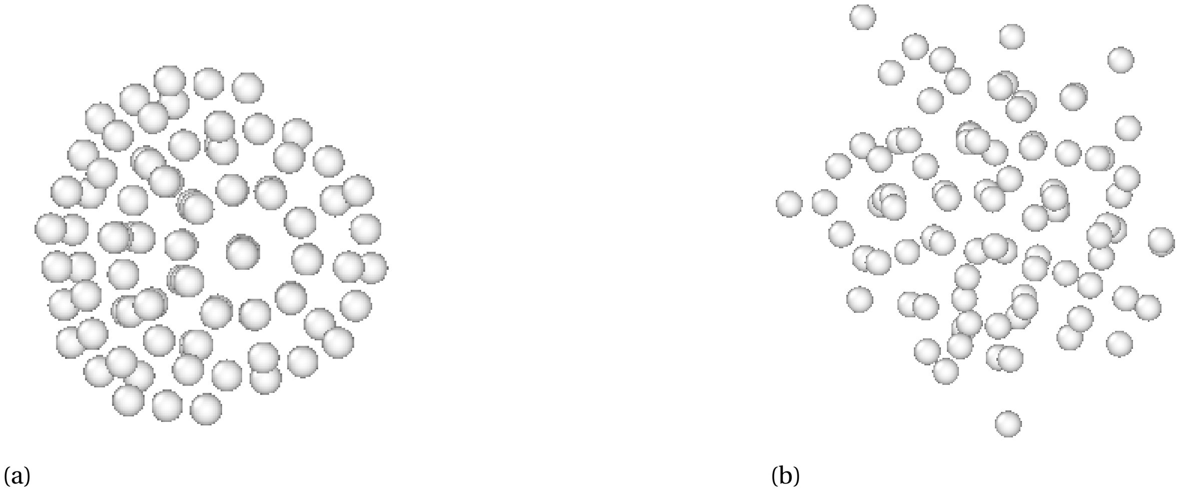


Figure 6.13: Different configurations assumed by a cluster of 91 particles. (a) reproduces the LJ global minimum stored in the Cambridge Energy Landscape Database, and (b) shows one of the minima structures found using OPP; the multiple circular structures can be clearly distinguished.

At first the blocks kept their shape but the line folded onto itself; once the blocks got agglomerated into a single cloud, they started to merged with each other until the borders of the single blocks were pretty much indistinguishable and the particles got arranged in a cluster actually quite similar to the Cambridge Cluster of 91 atoms (see Figure 6.13a), very close both in shape and energy (with a difference of ≈ 20 ϵ). The only significant difference noticed changing the disposition of the blocks when using LJ was the trend of the total energy of the minima found by the global optimization algorithm, but this can be easily explained: a cluster where the blocks are already agglomerated (3-dimensional configuration) will have a much more favourable energy than the cluster where the blocks are arranged in a line, therefore the first one will not have significant change of energies between the minima while the latter will exhibit a significant drop. The resulting stable configurations are however the same.

On the other hand, when using the Oscillating Pair Potential, the starting configuration played a major role for the results. In general, the most energetically-favourable structures were given by the starting configurations that were built with the icosahedral clusters of 13 atoms; in particular, the lowest energy was found to belong to the cluster with the blocks arrange in a plane. As the MD simulations disturbed the configurations more and more, the particles didn't spread out, but roughly maintained the type of configuration (in Figure 6.14 the line) initially given, and the blocks in the structure merged together and formed the circular arrangements that were already seen previously.

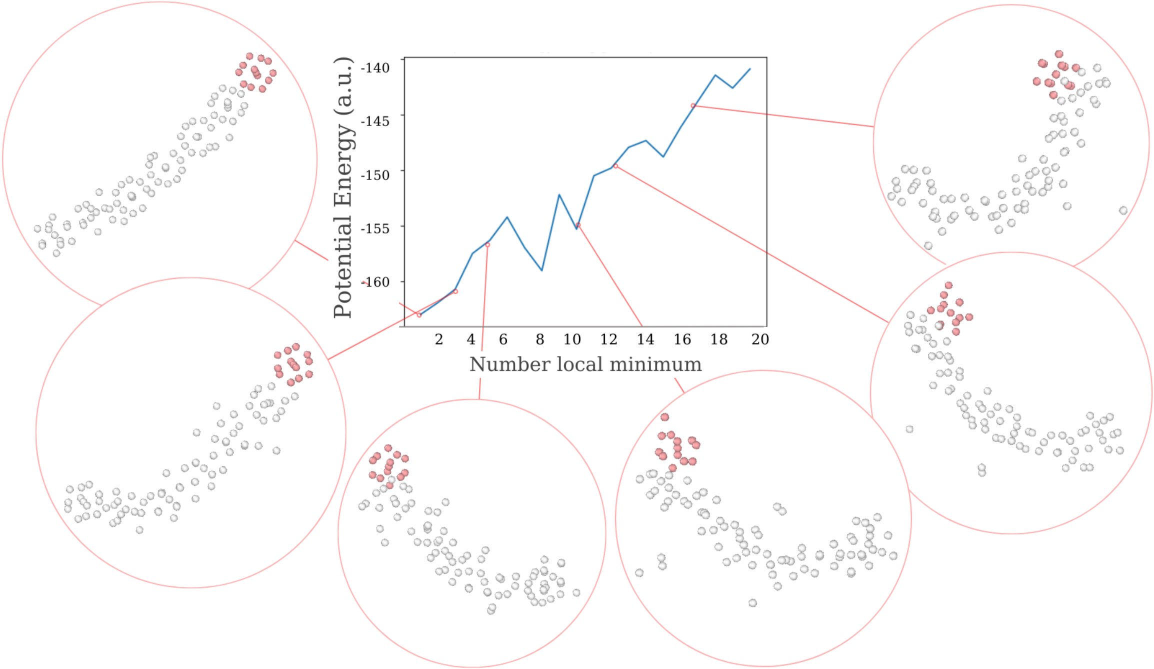


Figure 6.14: Structures of some minima when the line configuration of 7 blocks of 13 atoms (in total 91 atoms) was globally optimized using OPP.

It is therefore quite interesting to note how a system governed by the OPP potential tends to keep the general shape given initially to the cluster, but still forms circular substructures attached to each other.

7

Conclusions and Outlook

In this chapter the findings of this project are summarized, and some recommendations are provided to improve and expand the results of this research.

The purpose of this project was to improve the current understanding of the formation of quasicrystals. To this end, two methods were used: global geometry optimization and atomic simulations with the Oscillating Pair Potential. These methods had already been widely employed separately; global geometry optimization to identify the multiple minima configurations of a cluster of atoms and eventually its global minimum, and the Oscillating Pair Potential to model consistently the formation of IQCs structures in atomic clusters. However, they hadn't been yet combined together to investigate what are the energetically stable structures that can take shape into quasicrystalline materials and lead to the formation of non-periodic long-range order, proper of quasicrystals. Global geometry optimization allowed us to see different stable stages of the assembly of the atoms, and to distinguish among them the most energetically favourable configurations. The Oscillating Pair Potential proved to indeed be able to make clusters that tend to arrange into structures that exhibit some kind of order of multiple substructures, especially compared to the results yielded by a standard potential formula like the Lennard-Jones potential: while using LJ the particles always tried to keep the same distance between each other and to group around the same center, with OPP, no matter the starting configuration, the atoms always tended to settle on circular substructures attached to each other. Moreover, it was explored how engineering the structure of the cluster at the start would impact the results obtained. It was found that if a system governed by the OPP potential is manually built by arranging clusters of atoms in a certain shape, this tends to keep its general shape unaltered while still forming the circular substructures attached to each other. This could have important implications: if it is known that arranging groups of atoms with an icosahedral symmetry in certain shapes, the structure doesn't deviate significantly, this could potentially be used to make clusters self-assemble into quasicrystals.

There is however room for improvement. The results could not be validated using diffraction patterns, a widely known and used tool in the study of crystallography: the clusters investigated were too small to show a significant order when taking the FT of one projection of their structure. Another aspect to improve is the global optimization algorithm. First of all, for bigger clusters a much higher number of minima has to be obtained; the 20 now provided represents but a small fraction of all the minima configurations possible, and they don't deviate much from the initial configuration, from which then the results are very dependent. This can be done only once the memory-managing issue of the routine is solved; for now, this poses a hard limit on the amount of minima structures that can be investigated for one cluster.

For continuations after this study, it is recommended to first fix the aforementioned problems. Once the global optimization algorithm works efficiently, it would be useful to test for bigger clusters with sizes ranging $1000 < N < 5000$ and investigate whether recognizable symmetries would appear on the diffraction patterns of such structures. It would also be interesting to expand on the possibility to arrange atoms in pre-built blocks that are favourable to easily settle on a long-range ordered structure.

Bibliography

[1] C. Janot. *Quasicrystals, A Primer.* Oxford University Press, 1994.

[2] Stephen A. Nelson. Introduction and symmetry operations, 8 2013.

[3] Zhang Yue. A Rigorous Proof on the Crystallographic Restriction Theorem to Establish Human Being. *Budapest International Research and Critics Institute-Journal (BIRCI-Journal)*, 1(4):25–28, 12 2018.

[4] Dov Levine and Paul Joseph Steinhardt. Quasicrystals: A new class of ordered structures. *Physical Review Letters*, 53(26):2477–2480, 1984.

[5] R Penrose. The role of aesthetics in pure and applied mathematical research. *Bull.Inst.Math.Appl.*, 10:266–271, 1974.

[6] Michael Engel, Pablo F. Damasceno, Carolyn L. Phillips, and Sharon C. Glotzer. Computational self-assembly of a one-component icosahedral quasicrystal. *Nature Materials 2014 14:1*, 14(1):109–116, 12 2014.

[7] D. Shechtman, I. Blech, D. Gratias, and J. W. Cahn. Metallic phase with long-range orientational order and no translational symmetry. *Physical Review Letters*, 53(20):1951–1953, 11 1984.

[8] Sharon Glotzer. Quasicrystals: the thrill of the chase. *Nature*, 565(7738):156–158, 1 2019.

[9] D. Shechtman and I. A. Blech. The microstructure of rapidly solidified Al6Mn. *Metallurgical Transactions A*, 16(6):1005–1012, 6 1985.

[10] N. Rivier. Non-stick quasicrystalline coatings. *Journal of Non-Crystalline Solids*, 153-154(C):458–462, 2 1993.

[11] Adnene Sakly, Samuel Kenzari, David Bonina, Serge Corbel, and Vincent Fournée. A novel quasicrystal-resin composite for stereolithography. *Materials and Design*, 56:280–285, 2014.

[12] D Kieda, T Klein, and O Symko. Surgical Electric Blade Coated with Quasicrystalline Twin Films.

[13] Yasutaka Nagaoka, Hua Zhu, Dennis Eggert, and Ou Chen. Single-component quasicrystalline nanocrystal superlattices through flexible polygon tiling rule. *Science*, 362:1396–1400, 6 2018.

[14] Yasutaka Nagaoka, Jeremy Schneider, Hua Zhu, and Ou Chen. Quasicrystalline materials from non-atom building blocks. *Matter*, 6(1):30–58, 1 2023.

[15] Longfei Liu, Zhe Li, Yulin Li, and Chengde Mao. Rational Design and Self-Assembly of Two-Dimensional, Dodecagonal DNA Quasicrystals. *Journal of the American Chemical Society*, 141(10):4248–4251, 3 2019.

[16] Luca Bindi, Paul J. Steinhardt, Nan Yao, and Peter J. Lu. Icosahedrite, Al63Cu24Fe13, the first natural quasicrystal. *American Mineralogist*, 96(5-6):928–931, 5 2011.

[17] Eva G. Noya, Chak Kui Wong, Pablo Llombart, and Jonathan P.K. Doye. How to design an icosahedral quasicrystal through directional bonding. *Nature 2021 596:7872*, 596(7872):367–371, 8 2021.

[18] I Torrens. *Interatomic Potentials*. Elsevier Science, 2012.

[19] Marek Mihalkovič and C. L. Henley. Empirical oscillating potentials for alloys from ab initio fits and the prediction of quasicrystal-related structures in the Al-Cu-Sc system. *Physical Review B - Condensed Matter and Materials Physics*, 85(9):092102, 3 2012.

[20] Berni Julian Alder and Thomas Everett Wainwright. Phase transition for a hard sphere system. *The Journal of chemical physics*, 27(5):1208–1209, 1957.

[21] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation, From Algorithms to Applications*. 2002.

[22] Loup Verlet. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Physical Review*, 159(1):98, 7 1967.

[23] Molecular Dynamics (MD) - Compchems.

[24] Line Search Methods. *Numerical Optimization*, pages 34–63, 6 2006.

[25] Erik Bitzek, Pekka Koskinen, Franz Gähler, Michael Moseler, and Peter Gumbsch. Structural Relaxation Made Simple. *Phys. Rev. Lett.*, 97(17):170201, 10 2006.

[26] Peter Salamon, Paolo Sibani, and Richard Frost. *Facts, Conjectures, and Improvements for Simulated Annealing*. SIAM Monographs on Mathematical Modeling and Computation. 2002.

[27] David J. Wales and Jonathan P.K. Doye. Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. *Journal of Physical Chemistry A*, 101(28):5111–5116, 7 1997.

[28] Bemd Hartke. Global Geometry Optimization of Clusters Using Genetic Algorithms. Technical report, 1993.

[29] Ole Schütt. *Parallel Global Geometry Optimization of Molecular Clusters*. PhD thesis, Freie Universität Berlin, Zürich, 2014.

[30] Stefan Goedecker. Minima hopping: An efficient search method for the global minimum of the potential energy surface of complex molecular systems. *The Journal of Chemical Physics*, 120(21):9911–9917, 6 2004.

[31] Jeroen Vangindertael, Rafael Camacho, Wouter Sempels, Hideaki Mizuno, Peter Dedecker, and Kris Janssen. An introduction to optical super-resolution microscopy for the adventurous biologist. *Methods and Applications in Fluorescence*, 6, 7 2018.

[32] Joshua A. Anderson, Jens Glaser, and Sharon C. Glotzer. HOOMD-blue: A Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations. *Computational Materials Science*, 173, 2 2020.

[33] Alexander Stukowski. Visualization and analysis of atomistic simulation data with OVITO-the
}Open Visualization Tool. *MODELLING AND SIMULATION IN MATERIALS SCIENCE AND ENGINEERING*, 18(1), 1 2010.

[34] md.pair - HOOMD-blue 4.0.1 documentation.

[35] D. J. Wales, J. P. K. Doye, A. Dullweber, M. P. Hodges, F. Y. Naumkin, F. Calvo, J. Hernández-Rojas, and T. F. Middleton. The Cambridge Cluster Database.

[36] A. L. Mackay. A dense non-crystallographic packing of equal spheres. *Acta Crystallographica*, 15(9):916–918, 9 1962.

[37] Jonathan P.K. Doye, Mark A. Miller, and David J. Wales. Evolution of the Potential Energy Surface with Size for Lennard-Jones Clusters. *Journal of Chemical Physics*, 111(18):8417–8428, 11 1999.

[38] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

A

Code

A.1. Create Configurations in GSD Files from Cambridge Database

```python
import hoomd
import gsd.hoomd
import matplotlib
import numpy as np
import os
import sys
import glob
import matplotlib.pyplot as plt

for path in glob.glob(os.path.join(os.getcwd()+'/cambridge/txt-files/', '*new.TXT')):
    print(path)
    os.remove(path)
for path in glob.glob(os.path.join(os.getcwd()+'/cambridge/txt-files/', '*_old.TXT')):
    print(path)
    os.remove(path)
for path in glob.glob(os.path.join(os.getcwd()+'/cambridge/txt-files/', '*txt')):
    print(path)
    os.remove(path)

# load N_particles and trajectories to list position from text files
def load_traj(filename, start_dir):
    fd = open(start_dir+filename, "r")
    position = list()
    # for traj>150, the first line is empty so exclude it
    empty_line=False
    file = fd.readlines()
    N=len(file)
    if N>150:
        empty_line=True
    for line in file:
        if not empty_line:
            coord = [float(i) for i in line.split()]
            position.append(coord)
        empty_line=False
    return position

# function to create gsd files from txt files,
# giving parent directory and target directory
def create_gsd(start_dir, end_dir):
    if not os.path.isdir(end_dir):
        os.makedirs(end_dir)
    for filename in os.listdir(start_dir):
        # create snapshot
        traj = load_traj(filename, start_dir)
        N_particles = len(traj)
        snapshot = gsd.hoomd.Snapshot()
        snapshot.particles.N = N_particles
        snapshot.particles.position = traj[0:N_particles]
        dia = 1.9
        snapshot.particles.diameter = np.repeat(dia,N_particles)
        pos = snapshot.particles.position
        snapshot.particles.typeid = [0] * N_particles
        if N_particles<=50:
            L = 10
        else:
            L=N_particles/5
        snapshot.configuration.box = [L, L, L, 0, 0, 0] # box=10 too small for big numbers
        # write snapshot to cambridge-N=N_particles.gsd in folder cambgridge-gsd
        with gsd.hoomd.open(name=end_dir+str(filename)+'.gsd', mode='wb') as f:
            f.append(snapshot)
    return
```

```
# create directory to store cambridge gsd files
create_gsd(os.getcwd()+"/cambridge/txt-files/", os.getcwd()+"/cambridge/initial-configurations/")
```

A.2. Disturb Particles

```
import matplotlib
import numpy
import random

# check if particles overlap
def is_overlapping(elt, pos, dia):
    dia = 1.9 # if particles closer than 0.95 to each other, lj and opp have numerical problems
    crash = False
    for i in range(len(pos)):
        dist = numpy.sqrt((elt[0]-pos[i][0])**2+(elt[1]-pos[i][1])**2+(elt[2]-pos[i][2])**2)
        if dist<=0.5*dia: # radius particles
            crash = True
            break
        i=0
    return crash, i, dist

# move arbitrarily 1 particle given a snapshot of a simulation
def disturb_1particle(snapshot, dia):
    pos = snapshot.particles.position
    index1 = random.randint(0, pos.shape[0]-1)
    elt1 = pos[index1]
    pos = numpy.delete(pos, index1, 0)
    snapshot.particles.position[index1] += numpy.array([0.5,0.5,0.5]) # add 0.5 to each coordinate
    j=0
    overlaps, crash_i, dist = is_overlapping(snapshot.particles.position[index1], pos, dia)
    while overlaps: #enters if particle at index1 overlaps with some other
        snapshot.particles.position[index1][j%3] += 0.1 # add 0.1 to ONLY ONE coordinate
        j+=1
        overlaps, crash_i, dist = is_overlapping(snapshot.particles.position[index1], pos, dia)
```

```python
    return snapshot

# function that opens the file fn in the cambridge folder, disturbs it k times, and saves the new comfiguration in the disturbed-cambridge folder
def disturb_cambridge(fn, k):
    if not os.path.isdir(os.getcwd()+"/disturbed-cambridge/"):
        os.makedirs(os.getcwd()+"/disturbed-cambridge/")
    if not os.path.isdir(os.getcwd()+"/disturbed-cambridge/initial-configurations/"):
        os.makedirs(os.getcwd()+"/disturbed-cambridge/initial-configurations/")
    cpu = hoomd.device.CPU()
    sim = hoomd.Simulation(device=cpu, seed=1)
    sim.create_state_from_gsd(filename=os.getcwd()+"/cambridge/initial-configurations/"+str(fn)+".gsd")
    snapshot = sim.state.get_snapshot()
    for j in range(k): # move randomly k particles
        snapshot = disturb_1particle(snapshot, dia) # object of type hoomd.snapshot.Snapshot
    disturbed_snapshot = gsd.hoomd.Snapshot()
    disturbed_snapshot.particles.position = snapshot.particles.position
    disturbed_snapshot.particles.N = snapshot.particles.N
    disturbed_snapshot.particles.diameter = snapshot.particles.diameter
    disturbed_snapshot.particles.typeid = snapshot.particles.typeid
    disturbed_snapshot.configuration.box = [50,50,50,0,0,0]
    with gsd.hoomd.open(name=os.getcwd()+"/disturbed-cambridge/initial-configurations/"+str(fn)+'-dist-'+str(k)+'-times.gsd', mode='wb') as f:
        f.append(disturbed_snapshot)
    return
```

A.3. MD and Local Optimization Functions

```python
import itertools
import math
import gsd.hoomd
import hoomd
import matplotlib.pyplot as plt
import numpy
import os
import fresnel
import warnings
warnings.filterwarnings('ignore')

##############
# POTENTIALS #
##############
def opp_pot(range, k, phi):
    cell = hoomd.md.nlist.Cell(buffer=0)
    opp = hoomd.md.pair.OPP(nlist=cell, default_r_cut=range)
    opp.params[('A', 'A')] = {'C1': 1., 'C2': 1., 'eta1': 15,
                              'eta2': 3, 'k': k, 'phi': phi}
    opp.r_cut[('A', 'A')] = range
    return opp

# compute range OPP
def OPP_range(r, rmin, rmax, k, phi):
    cos = numpy.cos(k * (r - 1.25) - phi)
    sin = numpy.sin(k * (r - 1.25) - phi)
    V = numpy.power(r, -15) + cos * numpy.power(r, -3)
    F = 15.0 * numpy.power(r, -16) + 3.0 * cos * numpy.power(r, -4) + k * sin * numpy.power(r, -3)
    return (V, F)
# Determine the potential range by searching for extrema
def determineRange(k, phi):
    r = 0.5
    extremaNum = 0
    force1 = OPP_range(r, 0, 0, k, phi)[1]
    while (extremaNum < 6 and r < 5.0):
        r += 1e-5
        force2 = OPP_range(r, 0, 0, k, phi)[1]
        if (force1 * force2 < 0.0):
            extremaNum += 1
        force1 = force2
    return r

def lj_pot():
    cell = hoomd.md.nlist.Cell(buffer=0)
    lj = hoomd.md.pair.LJ(nlist=cell)
    lj.params[('A', 'A')] = dict(epsilon=1, sigma=1)
    lj.r_cut[('A', 'A')] = 2.5
    return lj

######################
```

```python
# MOLECULAR DYNAMICS #
######################
def setup_nvt(sim, pot, kT, tau):
    integrator = hoomd.md.Integrator(dt=0.005)
    #integrator.forces.append(lj_pot())
    integrator.forces.append(pot)
    nvt = hoomd.md.methods.NVT(kT=kT, filter=hoomd.filter.All(), tau=tau)
    integrator.methods.append(nvt)
    sim.operations.integrator = integrator
    return integrator

def setup_fire(sim,pot,dt):
    fire = hoomd.md.minimize.FIRE(dt=dt, force_tol=1e-2, angmom_tol=1e-1, energy_tol=1e-2, min_steps_conv=1)
    fire.forces.append(pot)
    fire.methods.append(hoomd.md.methods.NVE(hoomd.filter.All()))
    sim.operations.integrator = fire
    return fire

def run_MD(fn, pot, k, phi, range, kT, tau, nsteps, j):
    cpu = hoomd.device.CPU()
    sim = hoomd.Simulation(device=cpu, seed=1)
    sim.create_state_from_gsd(filename=fn)
    snapshot=sim.state.get_snapshot()
    N=snapshot.particles.N
    md_dir=os.getcwd()+'/global-opti/'+pot+'/'+str(N)+'/md/'
    if not os.path.isdir(md_dir):
        os.makedirs(md_dir)
    if pot=='opp-ld' or pot=='opp-id':
        pot_energy=opp_pot(range, k, phi)
    elif pot=='lj':
        pot_energy=lj_pot()
    setup_nvt(sim, pot_energy, kT, tau)
    sim.state.thermalize_particle_momenta(filter=hoomd.filter.All(), kT=1.5)
    thermodynamic_properties = hoomd.md.compute.ThermodynamicQuantities(filter=hoomd.filter.All())
    sim.operations.computes.append(thermodynamic_properties)
    sim.run(0)

    logger = hoomd.logging.Logger()
    logger.add(sim)
    logger.add(thermodynamic_properties)
    logger.add(pot_energy, quantities=['energies', 'forces'])
    gsd_writer = hoomd.write.GSD(filename=md_dir+'md-on-minimum-'+str(j)+'.gsd', trigger=hoomd.trigger.Periodic(1), mode='wb', filter=hoomd.filter.All(), log=logger)
    sim.operations.writers.append(gsd_writer)
    gsd_writer.log=logger
    sim.run(nsteps+1)
    del sim, gsd_writer, logger, thermodynamic_properties

    # plot energy during molecular dynamics
    #traj = gsd.hoomd.open(md_dir+'md-on-minimum-'+str(j)+'.gsd', 'rb')
    #timestep=[]
    #potential_energy=[]
    #for frame in traj:
    #    timestep.append(frame.configuration.step)
    #    potential_energy.append(
    #        frame.log['md/compute/ThermodynamicQuantities/potential_energy'][0])


    #fig, ax = plt.subplots()
    #ax.plot(timestep, potential_energy)
    #ax.set_title('potential energy during molecular dynamics using '+pot+' for '+str(N)+' particles')
    #ax.set_xlabel('timestep')
    #ax.set_ylabel('potential energy')
    #fig.savefig(md_dir+"potential-energy-md-on-minimum-"+str(j))
    #del timestep, walltime, potential_energy, traj
    return

##########################
# OPTIMIZE LOCAL GEOMETRY #
##########################
def optimize_local(filename, start_dir, pot, k, phi, range, j):
    cpu = hoomd.device.CPU()
    sim = hoomd.Simulation(device=cpu, seed=1)
    sim.create_state_from_gsd(start_dir+filename)
    init_snapshot = sim.state.get_snapshot()
    N=init_snapshot.particles.N
    minima_dir=os.getcwd()+"/global-opti/"+pot+'/'+str(N)+"/minima/"
    if not os.path.isdir(minima_dir):
        os.makedirs(minima_dir)
    if pot=='opp-ld' or pot=='opp-id':
        pot_energy=opp_pot(range, k, phi)
        dt=0.05
    elif pot=='lj':
        pot_energy=lj_pot()
        if N>150:
            dt=0.04
        else:
            dt=0.05
    fire = setup_fire(sim, pot_energy, dt) # set up fire as integrator of sim using potential 'pot'
thermodynamic_properties=hoomd.md.compute.ThermodynamicQuantities(filter=hoomd.filter.All())
    sim.operations.computes.append(thermodynamic_properties)
    logger = hoomd.logging.Logger()
    logger.add(sim)
    logger.add(thermodynamic_properties)
    logger.add(pot_energy, quantities=['energies', 'forces'])
    gsd_writer = hoomd.write.GSD(filename=minima_dir+'unsorted-minimum-'+str(j), trigger=hoomd.trigger.Periodic(1), mode='wb', filter=hoomd.filter.All(), log=logger)
    sim.operations.writers.append(gsd_writer)
    gsd_writer.log=logger
    i=0
    while not(fire.converged):
        i+=1
        sim.run(1)
    #print(N, "particles converged in ", i, "steps.")
    del sim, gsd_writer, logger, thermodynamic_properties
    traj = gsd.hoomd.open(minima_dir+'unsorted-minimum-'+str(j), 'rb')
    timestep=[]
    potential_energy=[]
    for frame in traj:
        timestep.append(frame.configuration.step)
        potential_energy.append(frame.log['md/compute/ThermodynamicQuantities/potential_energy'][0])
    fig, ax = plt.subplots()
    ax.plot(timestep, potential_energy)
    ax.set_title('potential energy during local optimization using '+pot+' for '+str(N)+' particles')
    ax.set_xlabel('timestep')
    ax.set_ylabel('potential energy')
    fig.savefig(minima_dir+"potential-energy-uns-minimum-"+str(j))
    del timestep, walltime
    del traj
    return potential_energy[-1]
```

- **OPP_range** and **determineRange** are compute the distance r_{cutoff} given certain values of k and ϕ.
- **opp_pot** and **lj_pot** create the pair forces of LJ and OPP and assign them to a given simulation.
- **setup_nvt** initializes and appends the integrator with the canonical ensemble NVT to the Simulation object in **run_MD**.
- **run_MD** runs and saves in a GSD file the trajectories of a Molecular Dynamics simulation starting from a configuration stored in a GSD file, given the file name, the potential with its parameters, and the parameters of MD: the temperature kT, the coupling τ and the number of steps *nsteps*.
- **setup_fire** initializes and appends the minimizer fire to the Simulation object in **optimize_local**. In this case the ensemble used is the microcanonical ensemble NVE.
- **optimize_local** returns the potential energy of the locally optimized geometry and in the process stores the trajectories of the particles in GSD files called unsorted-minimum-n, where n is the nth minimum found. The function takes as arguments the name of the file and the potential with its parameters.

A.4. Minima Hopping Algorithm

```
from md_and_opti_functions import *
from helper_functions import *
import sys
import time
import matplotlib.pyplot as plt
import numpy
import os
import warnings
warnings.filterwarnings('ignore')
# opp id parameters
k = 8.
phi = 0.5
# n particles and potential
```

```
N = int(float(sys.argv[1]))
pot = sys.argv[2]
start = time.time()
if pot == 'opp':
    range = determineRange(k, phi)
    print("range opp: ", range)
    if k == 6.25:
        pot = 'opp-ld'
    elif k == 8.:
        pot = 'opp-id'
kT = 1.
tau = 1.
nsteps = 250
md_dir = os.getcwd()+'/global-opti/'+pot+'/'+str(N)+'/md/'

if N > 50 and N <= 500:# the value of E_accept should increase
    n_min = 15         # with the increase of particles;
    E_accept = 1.      # for small clusters, =0.1
    hist_check = 1.
elif N > 500:          # hist_check is to check if the same minimum
    n_min = 20         # is already in history.
    E_accept = 5.      # minima have same energy <=> they are the same
    hist_check = 5.    # the value of hist_check should increase
else:                  # with n particles
    n_min = 15
    E_accept = 0.1
    hist_check = 0.5

# define initial configuration
file_cambridge = os.getcwd()+'/cambridge/initial-configurations/'+str(N)+'.gsd'
fn = file_cambridge

def get_dir_minimum(min):
    dir = os.getcwd()+'/global-opti/'+pot+'/'+str(N) + '/minima/unsorted-minimum-'+str(min)
    return dir

print("\n###########################################################")
print(" Executing GLOBAL OPTIMIZATION on a cluster of", N,
        "particles using the", pot, "potential.")
print("###########################################################\n")
a = 0
```

```
pot_arr = numpy.array([])
count = 0
while a < n_min:
    if a == 0:
        min_dir = os.getcwd()+"/global-opti/"+pot+'/'+str(N)+"/minima/"
        if pot == 'lj':
            run_MD(fn, pot, k, phi, range, kT, tau, 100, a)
            final_energy = optimize_local( 'md-on-minimum-'+str(a)+'.gsd', md_dir, pot, k, phi, range, a+1)
        else:
            final_energy = optimize_local(str(
                N)+'.gsd', os.getcwd()+'/cambridge/initial-configurations/', pot, k, phi, range, a+1)

        new_gsd = gsd.hoomd.open(
            os.getcwd()+"/global-opti/"+pot+'/'+str(N)+'/unsorted-all-minima', mode='wb')
    else:
        run_MD(get_dir_minimum(a), pot, k, phi, range, kT, tau, nsteps, a)
        fn = 'md-on-minimum-'+str(a)+'.gsd'
        final_energy = optimize_local(fn, md_dir, pot, k, phi, range, a+1)

    if a > 0 and numpy.amin(numpy.abs(pot_arr-final_energy)) < hist_check:
        kT = kT*1.1
    else:
        kT = kT/1.1
        if a == 0 or final_energy-pot_arr[-1] < E_accept:
            pot_arr = numpy.append(pot_arr, final_energy)
            E_accept = E_accept/1.1
            print("NEW MINIMUM FOUND! Amount minima found:", a+1)
            # append traj minimum found
            min_traj = gsd.hoomd.open(
                min_dir+'unsorted-minimum-'+str(a+1), mode='rb')[-1]
            new_gsd.append(min_traj)
            del min_traj
            a += 1
        else:
            E_accept = E_accept*1.1
    count += 1
    print(count)
```

```python
end = time.time()
dir = os.getcwd()+"/global-opti/"+pot+'/'+str(N)+"/minima/"

# sort minima from lowest to highest energy
sort = pot_arr.argsort()
for i in sort:
    os.rename(dir+'unsorted-minimum-'+str(sort[i]+1), dir+'minimum-'+str(i+1))
    os.rename(dir+'potential-energy-uns-minimum-' + str(sort[i]+1)+'.png', dir+'potential-energy-minimum-'+str(i+1)+'.png')

# create gsd files with snapshots all minima sorted
sorted_gsd = gsd.hoomd.open(
    os.getcwd()+"/global-opti/"+pot+'/'+str(N)+'/all-minima', mode='wb')
lst = numpy.arange(0, n_min, 1)
for h in lst:
    min_traj = gsd.hoomd.open(dir+'minimum-'+str(h+1), mode='rb')[-1]
    sorted_gsd.append(min_traj)
    del min_traj

# plot and save energies unsorted minima
x = numpy.arange(1, a+1, 1)
print(pot_arr)
fig, ax = plt.subplots()
ax.plot(x, pot_arr)
ax.set_title('potential energy during global optimization')
ax.set_xlabel('number of minimum')
ax.set_ylabel('final potential energy')
x1 = numpy.zeros(int(len(x)/2))
r = 0
while r < len(x1):
    x1[r] = 2*r
    r += 1
del r
ax.set_xticks(x1)
fig.savefig(os.getcwd()+"/global-opti/"+pot+'/' + str(N)+'/unsorted-potential-energy')

# plot and save energies sorted minima
sorted_pot = numpy.sort(pot_arr)
print(sorted_pot)
fig, ax = plt.subplots()
```

```
ax.plot(x, sorted_pot)
ax.set_title('potential energy during global optimization')
ax.set_xlabel('number of minimum')
ax.set_ylabel('final potential energy')
x1 = numpy.zeros(int(len(x)/2))
r = 0
while r < len(x1):
    x1[r] = 2*r
    r += 1
del r
ax.set_xticks(x1)
fig.savefig(os.getcwd()+"/global-opti/"+pot + '/'+str(N)+'/sorted-potential-energy')

# save energies to output file
numpy.savetxt(os.getcwd()+"/global-opti/"+pot+'/'+str(N) + '/energies.out', sorted_pot, delimiter=',')
```

A.5. Diffraction Patterns

```
import freud
import matplotlib.pyplot as plt
import rowan
import gsd.hoomd
import os
import sys

N = int(float(sys.argv[1]))
direction = str(sys.argv[2])
k = int(float(sys.argv[3])) # defines which minimum we take the diffraction pattern of
pot='opp-id'
if direction == 'x':
      arr = [1, 0, 0]
elif direction == 'y':
      arr = [0, 1, 0]
elif direction == 'z':
      arr = [0, 0, 1]
fn=os.getcwd()+'/global-opti/'+pot+'/'+str(N)+'/all-minima'
min_traj=gsd.hoomd.open(fn, mode='rb')[k]
diff_dir=os.getcwd()+'/global-opti/'+pot+'/'+str(N)+'/diffraction/'
if not os.path.isdir(diff_dir):
        os.makedirs(diff_dir)
dp = freud.diffraction.DiffractionPattern(grid_size=2048, output_size=2048)
fig, ax = plt.subplots(figsize=(4, 4), dpi=300)
view_orientation=rowan.vector_vector_rotation([0, 0, 1], arr)
dp.compute(min_traj, view_orientation=view_orientation)
dp.plot(ax, cmap='viridis') # bone
axis=rowan.rotate(view_orientation, [0, 0, 1])
print(axis)
title="Looking down the axis: ["+str(int(axis[0]))+', '+str(int(axis[1]))+', '+str(int(axis[2]))+']'
ax.set_title(title)
plt.savefig(diff_dir+str(N)+'-min-'+str(k)+'-'+direction+'.png')
```

A.6. Submission Script DelftBlue

```
#!/bin/bash
#SBATCH --job-name="global-opti"
#SBATCH --time=01:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=4G
#SBATCH --partition=compute
#SBATCH --account=Research-EEMCS-DIAM

# Load modules:
module load 2022r2
module load openmpi
module load miniconda3

# Activate conda, run job, deactivate conda
conda activate hoomd
srun python new-global-opti.py $1 $2 # 1: cluster size
conda deactivate                     # 2: potential
```

A.7. Create Multiple Clusters Configurations

```
import gsd.hoomd
import hoomd
import numpy
import numpy.linalg
```

```python
import os
import sys
import warnings
warnings.filterwarnings('ignore')

def max_value(list, i):
    return max([sublist[i] for sublist in list])

def min_value(list, i):
    return min([sublist[i] for sublist in list])

def shift(list, sign, i):
    if sign=='pos':
        return max_value(list, i)-min_value(list, i)+1.
    elif sign=='neg':
        return -(max_value(list, i)-min_value(list, i)+1.) # 1. is distance set between two clusters

def add_cluster(snapshot, sign, i, n, w, type_confi, save=False):
    # i = 0, 1, 2 to decide where to add cluster (0=x, 1=y, 2=z axis)
    # n = 1, 2, ... number of cluster to add
    N=snapshot.particles.N
    L=snapshot.configuration.box[0]
    dia=snapshot.particles.diameter[0]
    pos = snapshot.particles.position
    N_toadd = int(N/n)
    pos_copy=numpy.copy(pos[w*N_toadd:(w+1)*N_toadd])
    pos_toadd = numpy.reshape(pos_copy, (N_toadd*3))
    gap = shift(pos_copy, sign, i)
    for p in range(len(pos_toadd)):
        if i==p%3:
            pos_toadd[p]+=gap
    pos_toadd = numpy.append(pos, pos_toadd)
    pos_toadd = numpy.reshape(pos_toadd, (N+N_toadd, 3))

    new_snapshot = gsd.hoomd.Snapshot()
    new_snapshot.particles.position = pos_toadd
    new_snapshot.particles.N = N+N_toadd
    new_snapshot.particles.diameter = numpy.repeat(dia, N+N_toadd)
    new_typeid = numpy.append(snapshot.particles.typeid, numpy.ones(N_toadd))
    new_snapshot.particles.typeid = new_typeid
    new_snapshot.particles.types = ['B', 'A']
    new_dim=[L,L,L,0,0,0]
    # make box bigger in all directions - has to be square for diffraction patterns
    new_dim[:3]+=numpy.repeat(max(abs(max_value(pos_toadd, i)), abs(min_value(pos_toadd, i))), 3)
    new_snapshot.configuration.box = new_dim
    dir=os.getcwd()+'/multiple-clusters/'+str(N_toadd)+'/initial-configurations/'
    if not os.path.isdir(dir):
        os.makedirs(dir)
    if save==True:
        with gsd.hoomd.open(name=dir+str(n+1)+'-clusters-'+str(type_confi)+'.gsd', mode='wb') as f:
                f.append(new_snapshot)
    return new_snapshot

def create_multiple_clusters_confi(N_original, clusters, type_confi):
    cpu = hoomd.device.CPU()
    sim = hoomd.Simulation(device=cpu, seed=1)
    sim.create_state_from_gsd(filename=os.getcwd()+"/cambridge/initial-configurations/"+str(N_original)+".gsd")
    snapshot = sim.state.get_snapshot()
    sign='pos'
    r=0
    if type_confi=='line':
        while r<(clusters-1):
            save=False
            n=r+1
            if r<4 or r==clusters-2:
                save=True
            new_snapshot = add_cluster(snapshot, sign, 0, n, r, type_confi, save)
            snapshot = new_snapshot
            del new_snapshot
            r+=1
    elif type_confi=='2lines':
        while r<(clusters-1):
            save=False
            n=r+1
            i=int(r%2)
            w=2*int(r/2)
            if r<4 or r==clusters-2:
                save=True
            new_snapshot = add_cluster(snapshot, sign, i, n, w, type_confi, save)
            snapshot = new_snapshot
            del new_snapshot
            r+=1
    elif type_confi=='plane':
        l=int(numpy.sqrt(clusters)+0.5)
        while r<(clusters-1):
            save=False
            n=r+1
            if n%l==0:
                w=n-l
                i=1
            else:
                w=r
                i=0
            if r<4 or r==clusters-2:
                save=True
            new_snapshot = add_cluster(snapshot, sign, i, n, w, type_confi, save)
            snapshot = new_snapshot
            del new_snapshot
            r+=1
    elif type_confi=='3d':
        while r<(clusters-1):
            save=False
            if r>2:
                sign='neg'
            i=r%3
            n=r+1
            if r<4 or r==clusters-2:
                save=True
            new_snapshot = add_cluster(snapshot, sign, i, n, 0, type_confi, save)
            snapshot = new_snapshot
            del new_snapshot
            r+=1
    del cpu, sim
    print('Created', type_confi, 'of', N_original, 'x 1 to', clusters, 'atoms.')
    return

# uncomment the following lines if you want to create the multiple
```

```
    clusters' configurations for a given building block and amount of
    blocks

#init_N = int(float(sys.argv[1]))
#clusters = int(float(sys.argv[2]))
#create_multiple_clusters_confi(init_N, clusters, 'line')
#create_multiple_clusters_confi(init_N, clusters, '2lines')
#create_multiple_clusters_confi(init_N, clusters, 'plane')
#create_multiple_clusters_confi(init_N, clusters, '3d')
```

A.8. Minima Hopping for Multiple Clusters Configurations

```
    import gsd.hoomd
import hoomd
import matplotlib.pyplot as plt
import numpy
import os
import warnings
warnings.filterwarnings('ignore')
from helper_functions import *
from add_clusters import *
import sys

# parameters
init_N = int(float(sys.argv[1]))
clusters = int(float(sys.argv[2]))
type_confi=sys.argv[3]
pot = sys.argv[4]

print('arguments:',init_N, clusters, type_confi, pot)

# pot parameters
k=8.
phi=0.5
if pot=='opp':
    range=determineRange(k, phi)

def opp_pot(range, k, phi):
    cell = hoomd.md.nlist.Cell(buffer=0)
    opp = hoomd.md.pair.OPP(nlist=cell, default_r_cut=range)
    opp.params[('A', 'B')] = {'C1': 1., 'C2': 1., 'eta1': 15,
                        'eta2': 3, 'k': k, 'phi': phi}
    opp.r_cut[('A', 'B')] = range
    opp.params[('A', 'A')] = {'C1': 1., 'C2': 1., 'eta1': 15,
                             'eta2': 3, 'k': k, 'phi': phi}
    opp.r_cut[('A', 'A')] = range
    opp.params[('B', 'B')] = {'C1': 1., 'C2': 1., 'eta1': 15,
                             'eta2': 3, 'k': k, 'phi': phi}
    opp.r_cut[('B', 'B')] = range
    return opp

def lj_pot():
    cell = hoomd.md.nlist.Cell(buffer=0)
    lj = hoomd.md.pair.LJ(nlist=cell)
    lj.params[('A', 'B')] = dict(epsilon=1, sigma=1)
    lj.r_cut[('A', 'B')] = 2.5
    lj.params[('A', 'A')] = dict(epsilon=1, sigma=1)
    lj.r_cut[('A', 'A')] = 2.5
    lj.params[('B', 'B')] = dict(epsilon=1, sigma=1)
    lj.r_cut[('B', 'B')] = 2.5
    return lj

def setup_nvt(sim, pot, kT, tau):
    integrator = hoomd.md.Integrator(dt=0.005)
    integrator.forces.append(pot)
    nvt = hoomd.md.methods.NVT(kT=kT, filter=hoomd.filter.All(), tau=tau)
    integrator.methods.append(nvt)
    sim.operations.integrator = integrator
    return integrator

def setup_fire(sim,pot,dt):
    fire = hoomd.md.minimize.FIRE(dt=dt, force_tol=1e-3, angmom_tol=1e-2, energy_tol=1e-3, min_steps_conv=1)
    fire.forces.append(pot)
    fire.methods.append(hoomd.md.methods.NVE(hoomd.filter.All()))
    sim.operations.integrator = fire
    return fire

def run_md(original_N, clusters, pot, k, phi, range, kT, tau, nsteps, i, type_confi):
    cpu = hoomd.device.CPU()
    sim = hoomd.Simulation(device=cpu, seed=1)
    md_dir=os.getcwd()+'/multiple-clusters/'+str(original_N)+'/'+str(clusters)+'-clusters-'+str(type_confi)+'/'+pot+'/md/'
    if i==0:
        nsteps=50
        fn=os.getcwd()+'/multiple-clusters/'+str(original_N)+'/initial-configurations/'+str(clusters)+'-clusters-'+str(type_confi)+'.gsd'
    else:
        fn=os.getcwd()+'/multiple-clusters/'+str(original_N)+'/'+str(clusters)+'-clusters-'+str(type_confi)+'/'+pot+'/minima/uns-min-'+str(i)+'.gsd'
    sim.create_state_from_gsd(filename=fn)
    if pot=='lj':
        pot_energy=lj_pot()
    elif pot=='opp-ld' or pot=='opp-id' or pot=='opp':
        pot_energy=opp_pot(range, k, phi)
    setup_nvt(sim, pot_energy, kT, tau)
    sim.state.thermalize_particle_momenta(filter=hoomd.filter.All(), kT=kT)
    thermodynamic_properties = hoomd.md.compute.ThermodynamicQuantities(filter=hoomd.filter.All())
    sim.operations.computes.append(thermodynamic_properties)
    sim.run(0)
    logger = hoomd.logging.Logger()
    logger.add(sim)
    logger.add(thermodynamic_properties)
    logger.add(pot_energy, quantities=['energies', 'forces'])
    gsd_writer = hoomd.write.GSD(filename=md_dir+'md-on-min-'+str(i)+'.gsd',
                            trigger=hoomd.trigger.Periodic(1),
                            mode='wb',
                            filter=hoomd.filter.All(),
                            log=logger)

    sim.operations.writers.append(gsd_writer)
    gsd_writer.log=logger
    sim.run(nsteps+1)
    del sim, gsd_writer, logger, thermodynamic_properties
    return

def optimize_local(original_N, clusters, pot, k, phi, range, j, type_confi):
    cpu = hoomd.device.CPU()
    sim = hoomd.Simulation(device=cpu, seed=1)
    md_dir=os.getcwd()+'/multiple-clusters/'+str(original_N)+'/'+str(clusters)+'-clusters-'+str(type_confi)+'/'+pot+'/md/'
    fn='md-on-min-'+str(j)+'.gsd'
    sim.create_state_from_gsd(md_dir+fn)
    minima_dir=os.getcwd()+'/multiple-clusters/'+str(original_N)+'/'+
str(clusters)+'-clusters-'+str(type_confi)+'/'+pot+'/minima/'
    if not os.path.isdir(minima_dir):
        os.makedirs(minima_dir)
    if pot=='opp-ld' or pot=='opp-id' or pot=='opp':
        pot_energy=opp_pot(range, k, phi)
        dt=0.05
    elif pot=='lj':
        pot_energy=lj_pot()
        if original_N>150:
            dt=0.04
        else:
            dt=0.05
    fire = setup_fire(sim, pot_energy, dt) # set up fire as integrator
of sim using potential 'pot'
    thermodynamic_properties=hoomd.md.compute.ThermodynamicQuantities(
filter=hoomd.filter.All())
    sim.operations.computes.append(thermodynamic_properties)
    logger = hoomd.logging.Logger()
    logger.add(sim)
    logger.add(thermodynamic_properties)
    logger.add(pot_energy, quantities=['energies', 'forces'])
    gsd_writer = hoomd.write.GSD(filename=minima_dir+'uns-min-'+str(j
+1)+'.gsd', # not sorted yet by energy
                                trigger=hoomd.trigger.Periodic(1),
                                mode='wb',
                                filter=hoomd.filter.All(),
                                log=logger)
    sim.operations.writers.append(gsd_writer)
    gsd_writer.log=logger
    i=0
    while not(fire.converged):
        i+=1
        sim.run(1)
    del sim, gsd_writer, logger, thermodynamic_properties
    traj = gsd.hoomd.open(minima_dir+'uns-min-'+str(j+1)+'.gsd', 'rb')
    timestep=[]
    potential_energy=[]
    for frame in traj:
        timestep.append(frame.configuration.step)
        potential_energy.append(
            frame.log['md/compute/ThermodynamicQuantities/potential_energy'][0])
    fig, ax = plt.subplots()
    ax.plot(timestep, potential_energy)
    ax.set_title('potential energy during local optimization using '+pot+' for '+str(original_N)+'x'+str(clusters)+' particles')
    ax.set_xlabel('timestep')
    ax.set_ylabel('potential energy')
    fig.savefig(minima_dir+"potential-energy-uns-minimum-"+str(j))
    del timestep
    del traj
    return potential_energy[-1]

################################
# CREATE INITIAL CONFIGURATIONS #
################################
# dirs
init_dir=os.getcwd()+'/multiple-clusters/'+str(init_N)+'/initial-configurations/'
md_dir=os.getcwd()+'/multiple-clusters/'+str(init_N)+'/'+str(clusters)+'-clusters-'+str(type_confi)+'/'+pot+'/md/'
minima_dir=os.getcwd()+'/multiple-clusters/'+str(init_N)+'/'+str(clusters)+'-clusters-'+str(type_confi)+'/'+pot+'/minima/'
if not os.path.isdir(md_dir):
        os.makedirs(md_dir)
if not os.path.isdir(minima_dir):
        os.makedirs(minima_dir)
create_multiple_clusters_confi(init_N, clusters, type_confi)
tc=type_confi

#####################
# MOLECULAR DYNAMICS #
#####################
# md parameters
nsteps=250
tau=1.5
kT=1.
# on confi with #clusters clusters each of #init_N particles
print('run md')
run_md(init_N, clusters, pot, k, phi, range, kT, tau, nsteps, 0, tc)

######################
# GLOBAL OPTIMIZATION #
```

```
#######################
N = init_N*clusters
print('n particles:', N)
if N>50 and N<=500:
    n_min=15
    E_accept=1.
    hist_check=1.
elif N>500:
    n_min=20
    E_accept=5.
    hist_check=5.
else:
    n_min=15
    E_accept=1.
    hist_check=0.5
print('global parameters established.')

a=0
pot_arr=numpy.array([])
count=0
print("\n############################################################")
print(" Executing GLOBAL OPTIMIZATION on a cluster of", clusters, "x",
   init_N, "particles in", type_confi,"using the", pot, "potential.")
print("############################################################\n")
end_dir=os.getcwd()+'/multiple-clusters/'+str(init_N)+'/'+str(clusters)
   +'-clusters-'+str(type_confi)+'/'+pot
while a<n_min:
    if a==0:
        new_gsd=gsd.hoomd.open(end_dir+'/unsorted-all-minima', mode='wb
   ')
    run_md(init_N, clusters, pot, k, phi, range, kT, tau, nsteps, a, tc
   )
    final_energy=optimize_local(init_N, clusters, pot, k, phi, range, a
   , tc)
    if a>0 and numpy.amin(numpy.abs(pot_arr-final_energy))<hist_check:
        kT=kT*1.1
        print('minimum already in history')
        print('kT*1.1')
    else:
        kT=kT/1.1
        print('kT/1.1')
        if a==0 or final_energy-pot_arr[-1]<E_accept:
            pot_arr=numpy.append(pot_arr, final_energy)
```

```python
            E_accept=E_accept/1.1
            print("NEW MINIMUM FOUND! Amount minima found:", a+1)

            # append traj minimum found
            min_traj=gsd.hoomd.open(minima_dir+'uns-min-'+str(a+1)+'.gsd', mode='rb')[-1]
            new_gsd.append(min_traj)
            del min_traj
            a+=1
        else:
            print('energy too high, diff =', final_energy-pot_arr[-1])
            E_accept=E_accept*1.1
            print('E_accept*1.1')
    count+=1
    print(count)
# sort minima from lowest to highest energy
sort = pot_arr.argsort()
for i in sort:
    os.rename(minima_dir+'uns-min-'+str(sort[i]+1)+'.gsd', minima_dir+'min-'+str(i+1)+'.gsd')
# create gsd files with snapshots all minima sorted
sorted_gsd=gsd.hoomd.open(end_dir+'/all-minima', mode='wb')
lst=numpy.arange(0, n_min, 1)
for h in lst:
    min_traj=gsd.hoomd.open(minima_dir+'min-'+str(h+1)+'.gsd', mode='rb')[-1]
    sorted_gsd.append(min_traj)
    del min_traj
# plot energy minima
x=numpy.arange(1, a+1, 1)
print('unsorted energies:\n', pot_arr)
fig, ax = plt.subplots()
ax.plot(x, pot_arr)
ax.set_title('potential energy during global optimization')
ax.set_xlabel('number of minimum')
ax.set_ylabel('final potential energy')
x1=numpy.zeros(int(len(x)/2))
r=0
while r<len(x1):
    x1[r] = 2*r
    r+=1
del r
fig.savefig(end_dir+'/unsorted-potential-energy.png')
```

```
sorted_pot=numpy.sort(pot_arr)
print('sorted energies:\n', sorted_pot)
fig, ax = plt.subplots()
ax.plot(x, sorted_pot)
ax.set_title('potential energy during global optimization')
ax.set_xlabel('number of minimum')
ax.set_ylabel('final potential energy')
x1=numpy.zeros(int(len(x)/2))
r=0
while r<len(x1):
    x1[r] = 2*r
    r+=1
del r
fig.savefig(end_dir+'/sorted-potential-energy.png')
numpy.savetxt(end_dir+'/energies.out', sorted_pot, delimiter=',')
```