Parallel Global Geometry Optimization of Molecular Clusters

Diplomarbeit

zur Erlangung des Grades eines
Diplominformatikers

dem
Fachbereich Mathematik und Informatik
der Freien Universität Berlin
vorgelegt von

Ole Schütt
geboren in Braunschweig

Betreuer:
Prof. Dr. Frank Noé
Prof. Dr. Knut Reinert
Prof. Dr. Joost VandeVondele

Zürich 2014

Contents

1 Introduction

At the heart of computational chemistry lies the elucidation of structures. Knowing the exact positions of atoms in a molecular system is the prerequisite for any further investigation. This is most pronounced in computational biochemistry, where the *protein folding problem*, i.e. finding the natural structure of a protein through computer simulations is considered the "holy grail" of the field [1].

For many solid systems, the structure elucidation is equivalent to a global optimization problem of the potential energy with respect to the coordinates. In contrast to local optimization, a global optimization can only be guided by heuristics. Since the search space increases exponentially with the system size, this problem is NP hard [2].

Based on different heuristics a number of global search algorithms have been proposed. Unfortunately, these algorithms are not yet widely available in established chemistry software packages. Only these large software packages allow to easily combine numerous simulation techniques as it's necessary for everyday usage.

The goal of this thesis is to extend the popular software package CP2K with the functionality for performing parallel global geometry optimization.

Global optimization algorithms are typically designed for serial execution. However, serial computers have become irrelevant due to the physical limitations on clock speed [3]. The full power of today's super computers can only be leveraged by utilizing thousands of CPU cores in parallel [4]. A parallelized implementation is therefore inevitable.

CP2K is a software package for performing atomistic and molecular simulations [5]. It is an open source program that has been under development for more than ten years, and contains now over 1 million lines of Fortran 95 code. It is freely available under the General Public License and is used by many research groups. CP2K has been designed from the ground up as a parallel program, targeting current super computer architectures by using the Message Passing Interface standard.

Fortunately, all global optimization methods follow a common working-principle: They iteratively explore the configuration space by successively calculating the potential energy for new configurations. This allows to factor out common functionality into a framework. For the asynchronous handling of multiple parallel energy calculations, a master/worker software architecture is most suitable. For this purpose, a novel swarm-framework was implemented in CP2K. It takes care of most of the technical and boilerplate work and simplifies the implementation of the actual optimization algorithm considerably.

On top of the swarm-framework two optimization algorithms were implemented: Minima Hopping and Minima Crawling.

The structure of this thesis is as follows:

Chapter 2 gives a general introduction to the problem of global geometry optimization. The established benchmark system of Lennard-Jones clusters serves as an example.

As part of this thesis the Minima Hopping optimization method was implemented. It was proposed by Stefan Goedecker in 2004 and it is well suited for optimizing molecular clusters [6]. The method and the techniques it utilizes are presented in Chapter 3 and 4.

Furthermore, a novel optimization scheme, named *Minima Crawling*, was developed and implemented. It shows superior parallel performance over the Minima Hopping scheme and is described in Chapter 5.

The swarm-framework and other implementation details are covered in Chapter 6.

Eventually, in Chapter 7 the implemented global optimization schemes are compared using the established benchmark system of a Lennard-Jones cluster with 38 atoms.

2 Global Geometry Optimization

The global geometry optimization of molecular systems refers to the problem of finding the configuration of atoms, which yields the lowest possible potential energy. A *configuration* in this context refers to the set of coordinates of all atoms in the molecular system.

The field of computational chemistry has developed a wealth of methods to calculate the potential energy of a molecular system. Whereas some of these methods provide extremely accurate results, they come at tremendous computational costs. The other extreme are methods which capture the chemical reality only in an approximate and empirical way, but their costs are many orders of magnitude lower.

For the study and development of global geometry optimization algorithms, chemical accuracy is of less importance as long as the typical structure of the optimization problem remains realistic. One model that fulfills these requirements is the Lennard-Jones potential. It is the de facto standard system used for benchmarking global geometry optimization algorithms. In the following, the Lennard-Jones potential is introduced and it is used to sketch some of the main concepts of global geometry optimization.

2.1 Lennard-Jones Potential

The *Lennard-Jones potential* (LJ) is a simple pair potential, which models the interaction between two uncharged atoms. A pair potential $V(r)$ gives the potential energy of two particles with respect to their distance r. The LJ potential was first proposed by John Lennard-Jones in 1924 [7] and is defined as:

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] .$$

The LJ potential has two parameters: The σ parameter determines the equilibrium distance between the atoms, which is $2^{1/6}\sigma$. The ϵ parameter determines the energy at the equilibrium distance, and hence the depth of the potential well. The potential consists of an attractive and a repulsive term. Figure 2.1 shows these two terms together with the resulting LJ potential.

The r^{-6} term describes the attractive long-range interaction between the atoms, which are called *van der Waals* or *London dispersion* force [8]. They originate from instantaneous dipole-induced dipole forces. The long range field of an electric dipole decays with r^{-3}, which leads to the r^{-6} dependency of the dipole-induced dipole interaction.

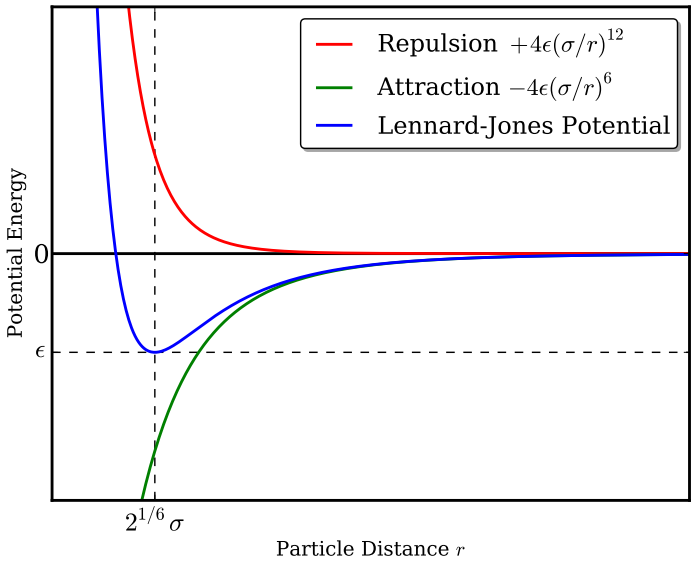


Figure 2.1: The Lennard-Jones Potential and its two constituent terms.

The r^{-12} term describes the short range repulsions, also called *Pauli repulsion* or *exchange interaction*, between two atoms. It is a purely quantum mechanical effect due to the fermionic nature of electrons. Fermions have to fulfill the *Pauli exclusion principle*, which states that two electrons must not occupy the same orbital [9]. When two atoms are brought in close contact, their electrons are forced onto higher orbitals in order to avoid occupying the same orbital. These excitations require energy, which leads to the repulsive force. The underlying quantum theories suggest that the Pauli repulsion depends exponentially on the distances, as pointed out by Buckingham [10]. However, the r^{-12} approximation used in the LJ potential is usually preferred, because it can be easily computed as the square of r^{-6}.

It should be emphasized that the LJ potential is merely an approximation and that more accurate models exist. However, due to its computational simplicity it is still used extensively in computer simulations.

2.2 Characterizing Potential Energy Surfaces

In the following a system consisting of N atoms is considered. All the atoms interact via the LJ potential with each other. The total energy E of such a system is given as the sum of the individual pair potential contributions:

$$E(\mathbf{r}_1, \ldots, \mathbf{r}_N) = \sum_{i=1}^{N} \sum_{j=1}^{i-1} V(|\mathbf{r}_i - \mathbf{r}_j|)\,.$$

The total energy depends on the positions of all particles and is also called a *Potential Energy Surface* (PES) defined in the *configuration space*. Since the PES is a high dimensional

function, it can only be visualized indirectly. In this regard, a very helpful concept is the *basin of attraction*. Each local minimum has an associated basin, which is defined as the region wherefrom a local optimization would be attracted towards the minimum. This is illustrated in Figure 2.2 for the one-dimensional case.

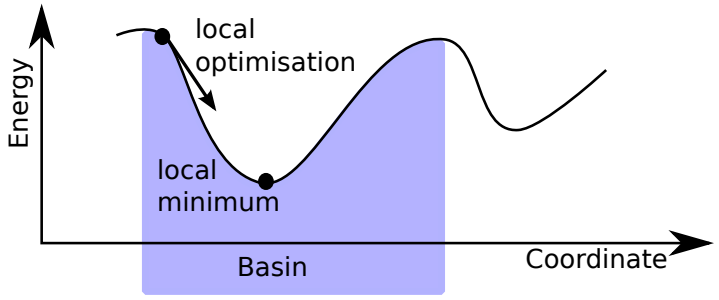


Figure 2.2: Illustration of a basin of attraction within a PES.

The basins are separated from each other by energy barriers of different heights. By choosing a threshold energy E_{T}, the basins can be further grouped together into disjoint sets called *super-basins*. A super-basin is defined in such a way that between each of its basins there exists a connecting path, which never crosses an energy barrier higher than the threshold energy E_{T}.

For a very high E_{T} there exists only a single super-basin. As the threshold energy is lowered, the super-basins split up into smaller super-basins. By performing the super-basin analysis for a series of decreasing threshold energies, a so-called *disconnectivity graph* can be created. These graphs were first introduced by Becker and Karplus [11]. In such a disconnectivity graph, a node represents a super-basin at a certain energy level. The edges connect nodes of adjacent energy levels and represent the splitting of super-basin towards a lower energy. In real systems the threshold energy could be provided by a heat bath with a certain temperature. Hence, the graphs show which regions of the configuration space are connected at a given temperature.

Figure 2.3 shows three examples of a one dimensional PES and their corresponding disconnectivity graphs for five energy levels. Here, a key concept is that of a *funnel*, which was originally developed in the protein folding community [12]. A funnel is a set of downhill pathways that converge towards a single low-energy minimum. Figure 2.3a shows an example of a PES with a single funnel. Finding the global minimum in such a system is rather easy. Figure 2.3b shows a PES with three main funnels. Such a system is much harder to optimize, because at high energies the "right" funnel has to be entered. The PES shown in Figure 2.3c has many funnels, which is characteristic for amorphous materials such as glass [13].

2.3 Lennard-Jones Cluster

Much of the initial interest in LJ clusters was motivated by the need to calculate nucleation rates for noble gases. However, the LJ potential has also become an established "testing

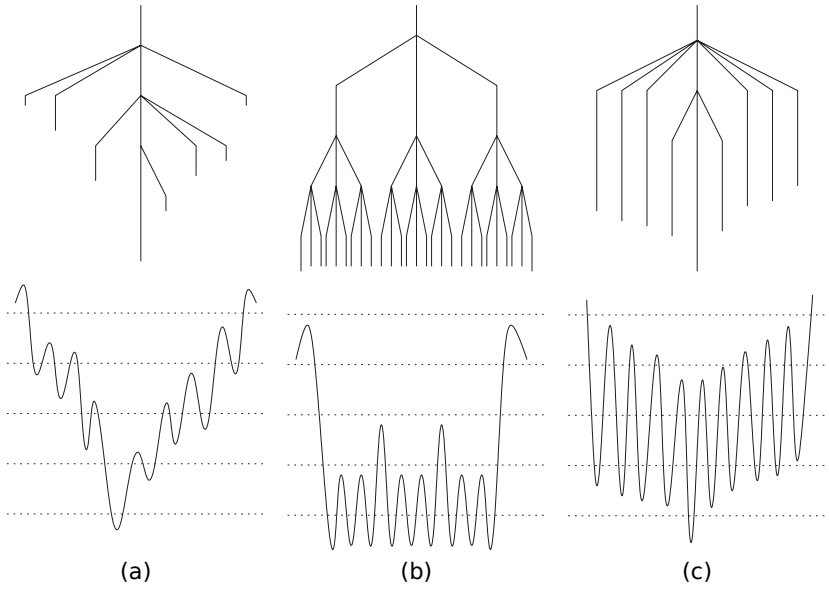


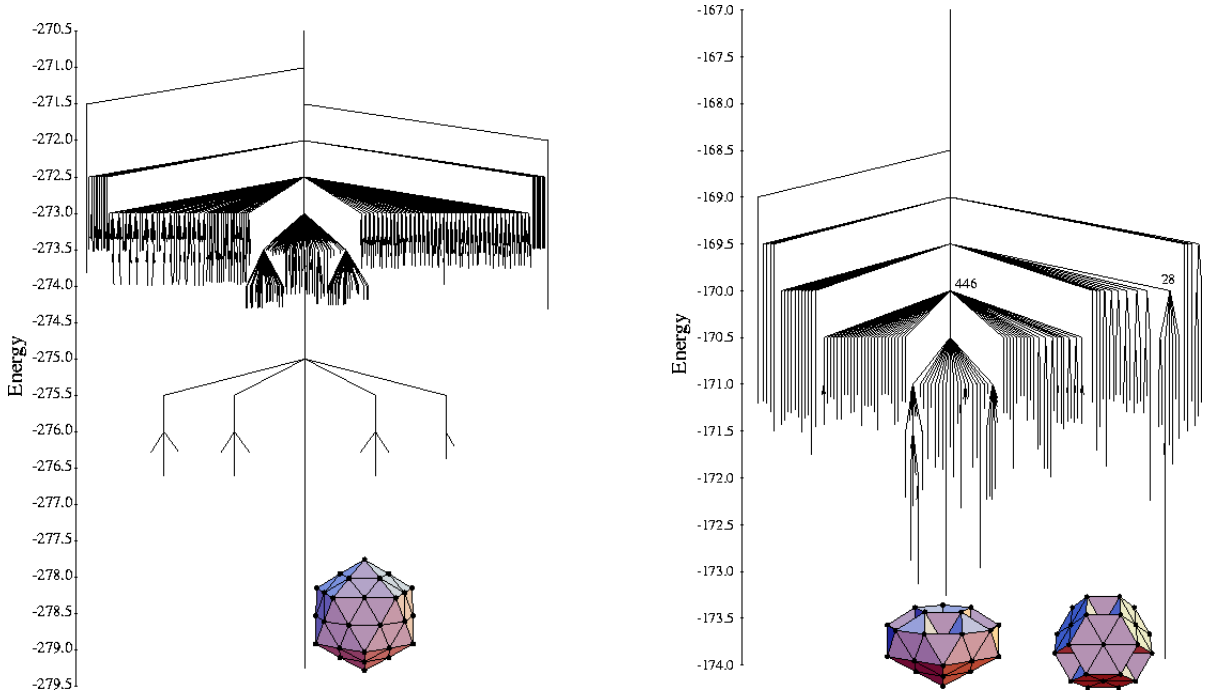
Figure 2.3: PES with corresponding disconnectivity graphs featuring different number of funnels. Graphics taken from Wales et al. [14]

ground" for global optimization algorithms. In a combined effort, likely candidates for the global minimum of LJ clusters with up to 1000 atoms have been found [15, 16, 17].

In general, finding the global minimum of a cluster is a NP hard problem [2]. However, in practice the level of difficulty depends very much on the structure of the PES. A prominent example is the Lennard-Jones cluster LJ_{55}, which consists of 55 particles. It has at least 10^{10} local minima, not counting permutational isomers. Nevertheless, in this specific case the global minimum is easy to locate, because the PES has only a single deep funnel as shown in Figure 2.4a. This is because 55 is a so called *magic number*, which allows to form the very symmetric *complete Mackay icosahedron* [15].

The Lennard-Jones cluster LJ_{38}, consisting of 38 atoms, is a prominent example for a double-funnel system [18]. Its global minimum structure is a *face-centered-cubic* (fcc) *truncated octahedron* with an energy of -173.928427ϵ. Its second lowest minimum is an *incomplete Mackay icosahedron* with an energy of -173.252378ϵ. Although these two lowest minima are less than 0.7ϵ apart, they are well separated in configuration space, because they are at the bottom of two different funnels (see Figure 2.4b).

When the LJ_{38} cluster is cooled down from a "liquidlike" state and its temperature drops below the melting point of around $0.18\epsilon k_B^{-1}$, it gets trapped in one of the two funnels. The optimization of the LJ_{38} is especially complicated, because it is very likely to enter the "wrong" funnel. The reason is that at high temperatures most configurations lead to the icosahedral funnel. In terms of thermodynamics, the configuration is favored *entropically*. Figure 2.5 shows the probability of the two structures over temperature. Near the melting-point there is still a high preference towards the icosahedral structure. The crossing point is at around $0.12\epsilon k_B^{-1}$, which is well below the melting-point where the system is already solidified and trapped in one of the funnels. [18]



(a) single-funnel LJ_{55} cluster (b) double-funnel LJ_{38} cluster

Figure 2.4: Disconnectivity graphs of two archetypal energy landscapes. Graphics taken from Wales et al., and Doye [14, 19]

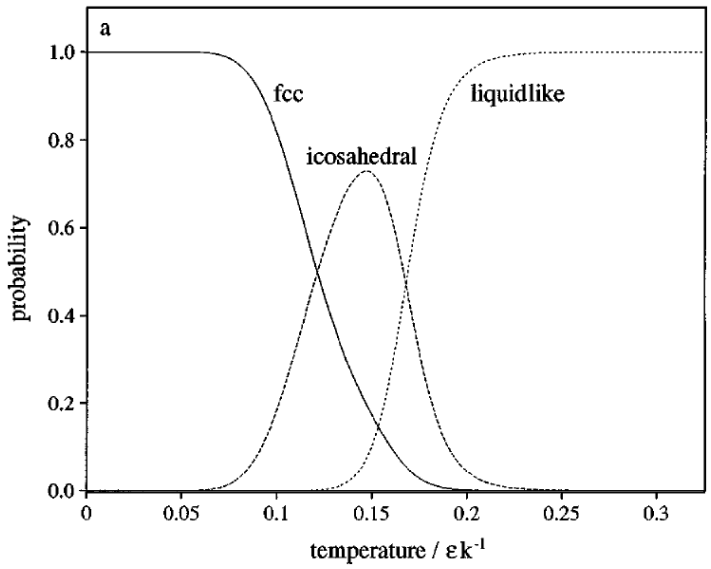


Figure 2.5: Probability of the LJ_{38} cluster to assume certain structures over temperature. Graphics taken from Doye and Wales [20]

The difficulties associated with the optimization of the LJ_{38} cluster have made it an established benchmark for global optimization algorithms. In this thesis, the LJ_{38} is used for all performance analysis.

3 Minima Hopping

In this chapter the Minima Hopping method for global geometry optimization is presented. The method was introduced in 2004 by Stefan Goedecker [6]. It has been successfully used to optimize silicon and gold clusters [21, 22], predict crystal structures [23], and to fold a small protein [24].

For this thesis Minima Hopping was chosen as the first scheme to be implemented. Among its advantages is the explicit usage of atomic forces to perform Molecular Dynamics. This suggests that the scheme is very well suited for the application to molecular systems. Furthermore, it only requires a small number of parameters compared to other schemes. This avoids the common problem of optimizing the parameters of the optimization scheme.

3.1 Remarks on Other Methods

Stefan Goedecker introduced Minima Hopping to resolve some of the weaknesses he saw in existing algorithms [6]:

Many algorithms for global optimization are based on thermodynamic principles. This includes standard algorithms such as simulated annealing [25], basin hopping [15], and multicanonical methods [26]. They explore the configuration space with a *Markov process* based on the *Metropolis algorithm* [27] with an acceptance probability given by the Boltzmann factor $\exp(-\Delta E/k_B T)$. For a sufficiently low temperature the ground state configuration will eventually dominate the sampling. However, this thermodynamic equilibration is most often prohibitively slow. The reason is the exponential decay of the acceptance probability for large energy increases ΔE. This effectively restricts the search to one funnel, because the climbing and crossing of high barriers, which separate different funnels, is very unlikely.

Another problem of many global optimization algorithms are repeated visits of certain regions in the configuration space. In an extreme case an optimization could get stuck by just jumping back and forth between two configurations. As a consequence, methods like *flooding* were developed [28, 29]. These methods artificially lift the PES for configurations that have been visited, which will eventually lift the PES of entire basins. However, the volume of a basin in configuration space can be enormous, which makes flooding inefficient. Furthermore, if a transition basin that connects two funnels is flooded, the optimization is slowed down.

Goedecker summarizes: "What is needed is a strategy that limits repeated visits, but does not penalize crossings through important transition basins." [6, p.9912]

Alternative methods for global optimization include basin hopping [15], gradient tabu search [30], genetic and evolutionary algorithms [31, 32], and biomimetic approaches, such as the artificial bee colony algorithm [33]. A very recent review on the topic was given by Heiles and Johnston [34].

3.2 Original Minima Hopping

The Minima hopping explores the PES iteratively. It has a current position x_0, which is advanced by performing promising escape jumps. The original algorithm consists of two nested loops, as illustrated in Figure 3.1. The inner loop performs *escape attempts* until an escape succeeds in discovering a new minimum x'. A successful escape is proposed as a possible jump to the outer loop. The outer loop accepts or rejects jumps based on their energy difference with repect to the current position. When a jump is accepted the current position is changed to the new minimum: $x_0 \leftarrow x'$.

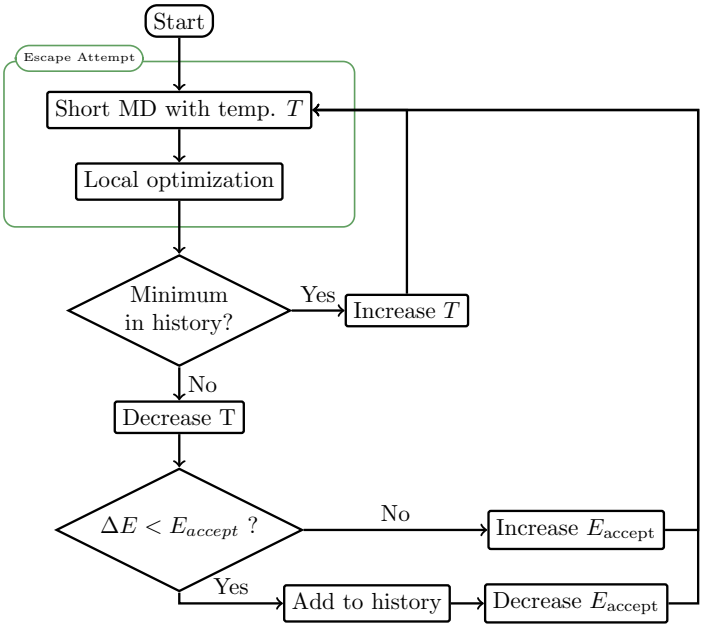


Figure 3.1: Flowchart of the original Minima Hopping scheme.

An escape attempt consists of a short Molecular Dynamics simulation followed by a local optimization: Starting from the current position x_0, the velocities are initialized according to a Boltzmann distribution of temperature T. During the MD the potential energy is observed. Once the potential energy has crossed a certain number of maxima and minima, typically three, the MD simulation is terminated. Afterward, the final configuration of the MD is used as a starting point for a local geometry optimization. The local optimization terminates when it has converged onto a local minimum x'.

If the found minimum is new, the escape attempt was successful. In order to determine if an escape attempt was successful, a database of previously found minima is maintained. In

the following this database is referred to as *history*. When an escape attempt is successful the temperature is decreased and the minimum is proposed to the outer loop as a possible escape jump. When an escape attempt is unsuccessful, because the minimum x' was already stored in the history, the temperature T is increased and a new escape attempt is made, starting from the original position x_0. By using increasingly higher temperatures, the MD will eventually manage to escape and discover a new minimum. This works even for deep funnels, e.g. by melting the molecular cluster.

The outer loop receives proposals for escape jumps from the inner loop. A jump is always accepted if its energy E' is lower than the energy of the current position E_0. Jumps that lead to an increase in energy are only accepted if the energy increase is below a certain threshold E_{accept}. Like the temperature T, the threshold E_{accept} is adjusted on-the-fly: When a jump gets accepted E_{accept} is decreased. When a jump is rejected E_{accept} is increased. This mechanism leads to a strong downwards preference, while also allowing for upward jumps after a funnel has been thoroughly explored.

The adjustment of the acceptance energy E_{accept} and the temperature T are carried out by multiplying or dividing with fixed factors, which are close to one. This introduces the two parameters α and β into the method, for which typical values are between 1.01 and 1.1 :

$$T_{\text{incr}} = \beta T_{\text{old}} \,, \qquad T_{\text{decr}} = \frac{1}{\beta} T_{\text{old}} \,, \qquad \beta > 1 \,,$$

$$E_{\text{incr}} = \alpha E_{\text{old}} \,, \qquad E_{\text{decr}} = \frac{1}{\alpha} E_{\text{old}} \,, \qquad \alpha > 1 \,.$$

3.3 Revised Minima Hopping

In the previous Section 3.2 the Minima Hopping scheme according to the original publication from 2004 was described. In the meantime further refinements were made. Professor Goedecker kindly provided his latest implementation of the algorithm [35]. For his research he uses a stand-alone Fortran program, which has the Lennard-Jones potential hard-coded. From this code the revised Minima Hopping algorithm was extracted (see Figure 3.2).

In the revised Minima Hopping scheme the two nested loops of the original scheme are fused into a single loop. Instead of the outer loop, which waits for the inner loop to make good escape proposals, there is now a variable x_{hop}, which stores the best possible escape found so far. This addresses a weakness of the original scheme related to increases of E_{accept}: In the original scheme a good escape proposal might at first get rejected because of E_{accept}. After a number of iterations E_{accept} is increased to the point where the good proposal would be accepted. However, the inner loop has to find the good escape again. In the meantime the inner loop has increased its temperature, which makes it unlikely to re-discover the low energy escape. Therefore, the original scheme sometimes missed a good escape route.

The revised scheme actually allows for hopping back and forth between known minima. This is a desired effect, because it allows for trying out different basins as starting point for an escape from a super-basin. Since rediscoveries lead to an increase of the temperature, eventually an escape will always be found.

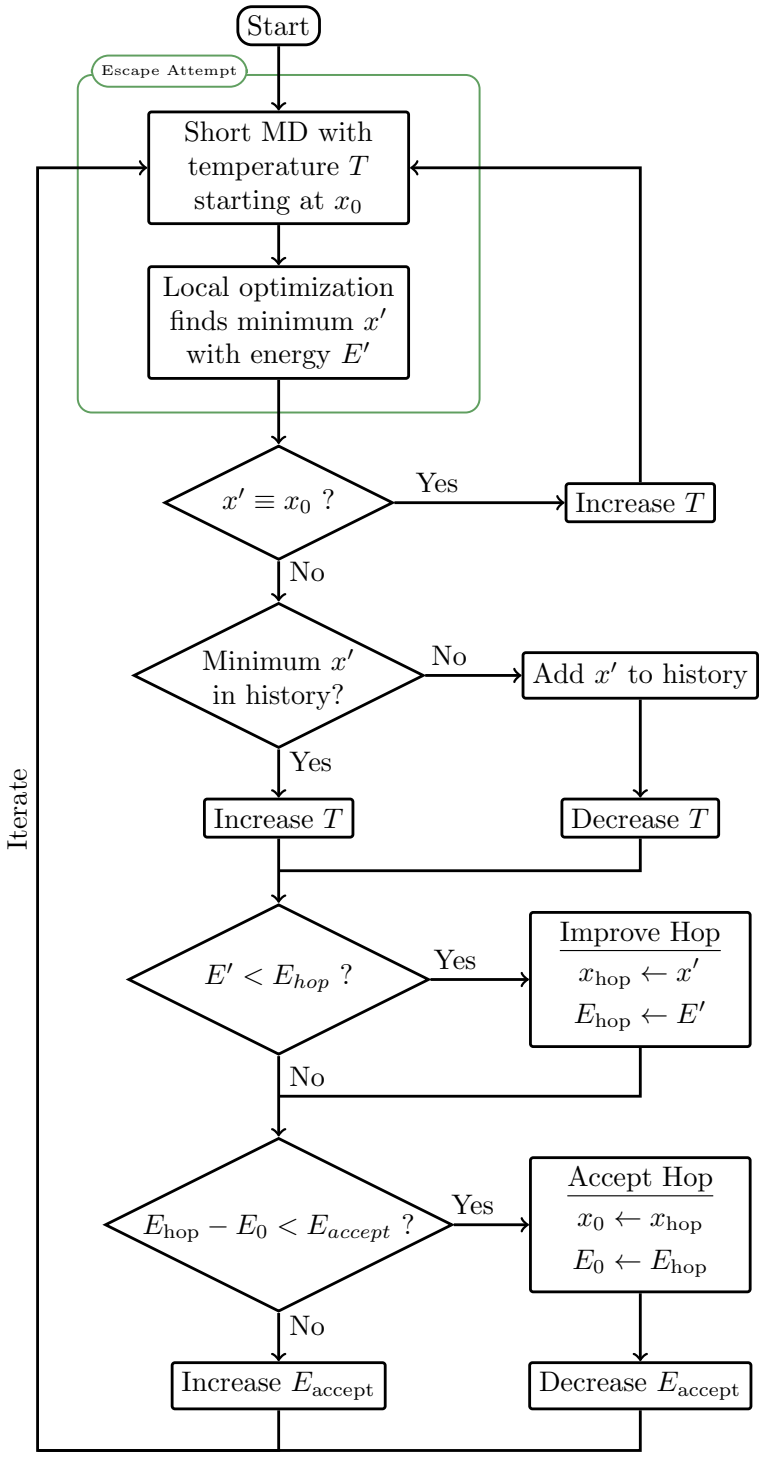


Figure 3.2: Flowchart of the revised Minima Hopping scheme as extracted from the Fortran code, which was kindly provided by Professor Goedecker [35].

3.4 Bell–Evans–Polanyi Principle

The on-the-fly adjustment of the temperature is essential for the success of the Minima Hopping scheme [6]. By keeping the temperature of the MD as low as possible, the search will mostly find escapes across low barriers. This is advantageous, because escapes across low barriers are more likely to lead into lower minima. In the field of chemistry this is well known as the *Bell-Evans-Polanyi principle* [36, 37]. It is an empirical observation made for many chemical reactions, which states that highly exothermic chemical reactions tend to have a low activation energy. Applied to the problem of global optimization this means that low energy minima are more likely to be found when crossing low energy barriers.

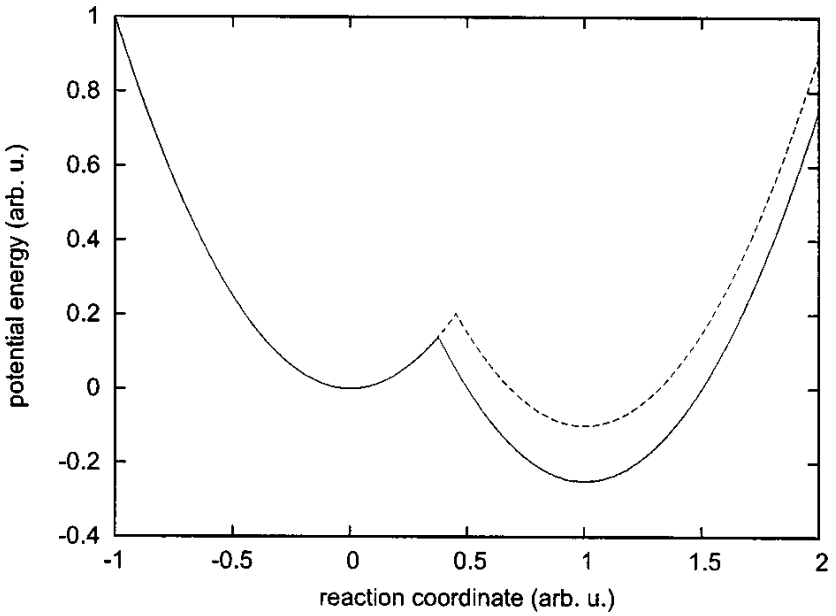


Figure 3.3: The Bell–Evans–Polanyi Principle: The deeper an adjacent minimum lies in energy, the lower is its transition barrier. Graphics taken from Goedecker [6]

The reason for this can be seen from Figure 3.3. It shows two neighboring minima, which are approximated with parabolas. For the right minimum two parabolas are drawn for two different energy levels. It shows that if the right minimum were higher in energy (dashed line) the transition barrier would also be higher in energy. An in-depth discussion of the topic was given by Roy et al. [38].

4 Escape Attempt Techniques

The Minima Hopping scheme makes use of a number of other simulation techniques. In this chapter these techniques are briefly introduced. Most of the techniques are needed to perform escape attempts. The calculation and comparison of descriptors is needed at the end of an escape attempt to decide whether it was successful or not.

4.1 Velocity Initialization

At the beginning of each MD simulation initial velocities are generated: A set of velocities is drawn from a *standard normal distribution* $\mathcal{N}(0,1)$ using a *random number generator*. The velocities of each atom are weighted according to its mass m_i:

$$\mathbf{v}_i' \sim \frac{1}{\sqrt{m_i}} \mathcal{N}(0,1)\,.$$

Then the rigid body motion is subtracted from the velocities. This includes the linear velocity of the *center of mass* $\mathbf{v}'_{\mathrm{com}}$ as well as angular velocities with respect to the center of mass $\vartheta'_{\mathrm{com}}$:

$$\mathbf{v}_i'' = \mathbf{v}_i' - \mathbf{v}'_{\mathrm{com}} - \vartheta'_{\mathrm{com}}\,.$$

Based on these velocities the kinetic energy is calculated:

$$E''_{\mathrm{kin}} = \sum_{i=1}^{N} \frac{1}{2} m_i |\mathbf{v}_i''|^2\,.$$

Eventually, the velocities are scaled to ensure that on average each degree of freedom has a kinetic energy of $k_{\mathrm{B}}T/2$:

$$\mathbf{v}_i = \alpha \mathbf{v}_i'' \qquad \text{with} \qquad \alpha = \sqrt{\frac{3N\,k_{\mathrm{B}}T}{2E''_{\mathrm{kin}}}}\,.$$

The scaled velocities $\mathbf{v}_i$ are then used for the initial time step of the MD simulation.

4.2 Velocity Softening

Originally, the velocities were chosen in a completely random fashion, as described in the previous Section 4.1. However, it is known that the low energy saddle points often lie at the end of low-curvature modes [38]. Henkelman and Jónsson proposed a *Dimer method* for finding saddle points, which only requires the first derivatives of the PES i.e. atomic forces [39]. Schönborn et al. adopted this method to beneficially bias the initial velocities used in the Minima Hopping scheme and named it *velocity softening* [40].

The velocity softening method finds the direction of such a low-curvature mode iteratively. The direction is given as the difference between the current position $\mathbf{x}$ and a test-point $\mathbf{y}_i$, which is updated in every iteration. The initial $\mathbf{y}_0$ is chosen at a fixed distance d in the direction of the original random velocity $\mathbf{v}$:

$$\mathbf{y}_0 = \mathbf{x} + d\frac{\mathbf{v}}{|\mathbf{v}|}\,.$$

In each iteration the force at the test-point $\mathbf{y}_i$ is calculated:

$$\mathbf{F}_i = \mathbf{F}(\mathbf{y}_i)\,.$$

Then, the force component, which is perpendicular to the current direction $\hat{\mathbf{N}}_i$, is calculated:

$$\mathbf{F}_i^{\perp} = \mathbf{F}_i - (\mathbf{F}_i \cdot \hat{\mathbf{N}}_i)\hat{\mathbf{N}}_i \qquad \text{where} \qquad \hat{\mathbf{N}}_i = \frac{\mathbf{y}_i - \mathbf{x}}{|\mathbf{y}_i - \mathbf{x}|}\,.$$

The test-point is then updated with the force component $\mathbf{F}_i^{\perp}$ using a fixed mixing-factor α:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \alpha\mathbf{F}_i^{\perp}\,.$$

After the last softening iteration was performed, the direction $\hat{\mathbf{N}}_i$ is used as input for the new velocities. Again, the rigid body motion has to be subtracted and they have to be scaled to match the requested temperature, as described in Section 4.1.

The velocity softening method will eventually converge when $\mathbf{F}_i^{\perp}$ becomes zero. This would remove all of the initial randomness and make the velocities deterministic. However, a certain amount of randomness is essential for the Minima Hopping scheme to work. Otherwise, the MD will always escape into the same soft mode direction and would not discover any other escape routes. Therefore, only a few softening iterations should be performed. Schönborn et al. reported that 20 softening iterations give the best results for Lennard-Jones clusters (see Section 7.2).

4.3 Molecular Dynamics

In this section *Newton's notation* for differentiation is used:

$$\dot{m} \equiv \frac{dm}{dt}\,, \qquad \dot{\mathbf{r}} \equiv \frac{d\mathbf{r}}{dt}\,, \qquad \ddot{\mathbf{r}} \equiv \frac{d^2\mathbf{r}}{dt^2}\,.$$

The goal of Molecular Dynamics simulations is to integrate *Newton's second law of motion*. This law relates the position of a particle $\mathbf{r}(t)$ at a time t to the force $\mathbf{F}$ that acts upon the particle. Under the assumption of constant mass $\dot{m} = 0$ it states:

$$\mathbf{F} = m\ddot{\mathbf{r}}\,.$$

The simplest way to numerically integrate Newton's second law is to iteratively propagate it in small time steps τ. An expression for a small time step can be obtained from a Taylor expansion. It gives the particle position at time $t + \tau$ in terms of its position $\mathbf{r}$, velocity $\dot{\mathbf{r}}$, and acceleration $\ddot{\mathbf{r}}$ at time t:

$$\mathbf{r}(t+\tau) = \mathbf{r}(t) + \tau\,\dot{\mathbf{r}}(t) + \frac{\tau^2}{2}\ddot{\mathbf{r}}(t) + \mathcal{O}(\tau^3)\,. \tag{4.1}$$

Alternatively, one can also start at $t+\tau$ and use a time-step of $-\tau$ to propagate backwards:

$$\mathbf{r}(t) = \mathbf{r}(t+\tau) - \tau\,\dot{\mathbf{r}}(t+\tau) + \frac{\tau^2}{2}\ddot{\mathbf{r}}(t+\tau) + \mathcal{O}(\tau^3)\,. \tag{4.2}$$

Adding the two previous equations (4.1) and (4.2) yields:

$$\dot{\mathbf{r}}(t+\tau) = \dot{\mathbf{r}}(t) + \frac{\tau}{2}\ddot{\mathbf{r}}(t) + \frac{\tau}{2}\ddot{\mathbf{r}}(t+\tau) + \mathcal{O}(\tau^2)\,. \tag{4.3}$$

Together the equations (4.1) and (4.3) define the *velocity Verlet algorithm* [41]. It has the benefits of being symplectic and time-reversible, which leads to a good conservation of energy and momentum. Furthermore, it minimizes round-off errors by avoiding the addition of small and large terms. Unlike the *Leapfrog algorithm*, velocities and positions are calculated for the same points in time. The position's error is of order 3, although only one force evaluation is required per time step. In combination, these features make the velocity Verlet algorithm the integrator of choice for Molecular Dynamics simulations.

For further readings about Molecular Dynamics and related techniques the book by Allen and Tildesley [42], as well as the book by Tuckerman [43] are recommended.

4.4 Cluster Defragmentation

During an MD simulations, it sometimes occures that one or more particles are ejected from the cluster. The subsequent local optimization does usually not correct for this, because the long range term of the Lennard-Jones potential decays very quickly with r^{-6}. Therefore, after every MD simulation the cluster has to be explicitly checked for fragmentation and if necessary merged back together.

For the fragmentation analysis, a graph is created which reflects the cluster's topology. The nodes of this graph are the atoms. Two atoms are connected with an edge if they are closer than a given threshold. If the graph has more than one *connected component*, the cluster got fragmented. The smaller fragments are then translated towards the largest fragment such that their outermost atoms are just below the threshold distance. From there on the subsequent local optimization will compact the fragments further.

4.5 Local Optimization

The goal of a local optimization is to find the nearest local minimum on the PES. Typically, the optimization is performed iteratively: The current position is advanced stepwise such that the potential energy is lowered until it's converged onto the local minimum. In each step a search direction p_k and a step length α_k has to be chosen:

$$x_{k+1} = x_k + \alpha_k p_k \,.$$

Once a direction is chosen, the optimal step length α_k can be found through a line search within the one-dimensional subspace. A popular method for line search is the *golden section search* [44]. The method works by successively approaching the minimum from both sides with a point. A third middle point ensures that the minimum remains between the two outer points. In each step the interval is narrowed by moving one of the points. The distances between the three points are always kept in a golden ratio, which guarantees good convergence even for the worst case. The search terminates when the distance between the outer points drops below a certain threshold.

A very popular scheme for choosing the search direction p_k is the *Conjugate Gradient method* (CG). It chooses the directions as the gradient at the current position plus a correction based on the direction from the previous step:

$$p_{k+1} = -\nabla f(x_{k+1}) + \beta_{k+1} p_k \,.$$

For the pre-factor β multiple options exists, a popular choice is the following, called *Fletcher-Reeves method* [45]:

$$\beta_{k+1} = \frac{|\nabla f(x_{k+1})|^2}{|\nabla f(x_k)|^2} \,.$$

For the special case of $f(x)$ being a strongly convex quadratic function, and given that the line search is exact, it can be shown that the conjugate gradient method converges in at most n steps, where n is the dimension of the configuration space. In practice these conditions are hardly ever met. Nevertheless, the convergence is guaranteed if the line search provides sufficient accuracy to satisfy the *strong Wolfe conditions* [46, 47]:

$$\begin{aligned} f(x_k + \alpha_k p_k) &\leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T \cdot p_k \,, \\ |\nabla f(x_k + \alpha_k p_k)^T \cdot p_k| &\leq -c_2 \nabla f(x_k)^T \cdot p_k \,, \\ \text{with} \quad 0 < c_1 < c_2 &< \frac{1}{2} \,. \end{aligned}$$

Another very popular class of local optimization algorithms are *Quasi-Newton methods*. They work by maintaining an approximation B_k of the Hessian matrix $\nabla^2 f$. In each step B_k is updated to incorporate the additional information gained by calculating the gradient at a new position. One of the most popular formulas for updating B_k is the *BFGS formula*, which was named after its inventors, Broyden [48], Fletcher [49], Goldfarb [50], and Shanno [51]:

$$s_k = x_{k+1} - x_k \,, \qquad y_k = \nabla f(x_{k+1}) - \nabla f(x_k) \,, \qquad B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \,.$$

From the approximated Hessian B_k the new direction of a search step is obtained via:

$$p_k = -B_k^{-1} \cdot \nabla f(x_k)\,. \tag{4.4}$$

Normally, the Quasi-Newton optimizers are considered superior over the conjugate gradient method. However, for the cheap Lennard Jones potential it turned out that the additional computational cost for the matrix inversion of B_k in (4.4) outweighs the benefits. Therefore, after an initial phase of experimentation the CG method was used in this thesis.

For proofs and an in-depth discussion of numerical optimization the book by Nocedal and Wright [52] is recommended.

4.6 Minima Descriptors

The identification of previously visited minima is essential for the Minima Hopping scheme. It requires that a database of the search history is maintained during the optimization. When this database is queried it should recognize the contained configurations 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.

In the case of classical force fields the potential energy can serve as a descriptor, because it can be determined with sufficient accuracy to be considered as unique. However, with other methods, e.g. *Density Functional Theory* [53, 54], this is not the case, because they contain more numeric noise. For these applications Goedecker suggested to extend the descriptor with the sorted list of all inter-atomic distances [55].

For this thesis another descriptor is used, which was obtained from the code provided by Professor Goedecker [35]. It uses the eigenvalues of the following symmetric matrix A as descriptor, where $\mathbf{r}_i$ denotes the position of the i-th atom:

$$A_{ij} = e^{-|\mathbf{r}_i - \mathbf{r}_j|^2/2}\,.$$

Two descriptors are compared based on their Euclidean distance. If the distance is smaller than a given threshold, the two configurations are considered equal. If multiple configurations fall below the threshold, the closest match is selected.

Over the course of a longer optimization run, the history can be filled with a large number of visited minima. In order to make the look-up efficient, a simple two step strategy is used: At first, the entries are filtered based on their energy. Only those entries are selected, which lie within the threshold-window. This reduces the number of candidates to just a few entries. For these remaining entries all components of the descriptor have to be compared. In order to speed up the initial energy-filtering, a sorted list of the minima is maintained. The elements are located within the sorted list by using *interpolation search* [56, p.419].

5 Minima Crawling

The Minima Hopping scheme was developed as a serial algorithm. It can not be parallelized in a straight forward way, because of data dependencies between the iterations.

Schönborn et al. suggest to parallelize the Minima Hopping scheme by letting multiple workers share a single history of visited minima. The idea is that due to the feedback mechanism and the common history an overlap of search area will be penalized. They claim an "almost linear speedup in runtime" can be achieved [40, p.5].

This thesis' benchmarks (see Chapter 7) show that the runtime does indeed reduce significantly, when multiple Minima Hopping workers with a shared history are used. However, basically the same performance can be observed for an *embarrassing parallelization*. In an embarrassing parallelization multiple completely independent workers are launched. This shows that the shared history scheme is not able to take advantage of the information exchange between the workers.

Given that the master has the collective information of all workers, it should be able to make much better decisions than a single worker with its smaller amount of information. Therefore, it should be possible to design a parallelization scheme for Minima Hopping that performs significantly better than an embarrassing parallelization. The Minima Crawling scheme is the attempt made in this thesis to develop such an improved parallelization scheme.

The history sharing scheme seems to suffer from two weaknesses:

1. The information exchange between the workers is too little: The course of one worker is only affected by another worker when they both happen to visit the same minimum, which is a rare event.

2. The state of a Minima Hopping worker does not solely consists of its minima history. It also includes the current temperature, acceptance energy, and hopping candidate. By affecting only the minima history, the scheme leads to inconsistencies within this state.

Furthermore, the poor performance of the history sharing scheme could also be the result of subtle effects such as two workers systematically interfering with each others progress. Unfortunately, it is very hard to investigate these phenomenas thoroughly. After all, a run with multiple workers constitutes a highly parallelized program with many state-variables, a fair amount of random behavior and race conditions.

The idea of the Minima Crawling scheme is to simplify the situation by making the workers state-less and instead gathering all available information in a central data structure. The key operation of performing an escape attempt is adopted without change from the Minima Hopping scheme. All encountered minima are stored in a central history together with additional status information.

An overview of the Minima Crawling scheme is shown by the flowchart in Figure 5.1. For each escape attempt a minimum from the history is chosen based on a ranking. The chosen minimum will be referred to as the *starting minimum*. Then an appropriate temperature is drawn from the starting minimum's temperature distribution. The starting minimum's configuration and the drawn temperature are used to perform an escape attempt. After the escape attempt, the record of the starting minimum will be updated with the newly gained information.

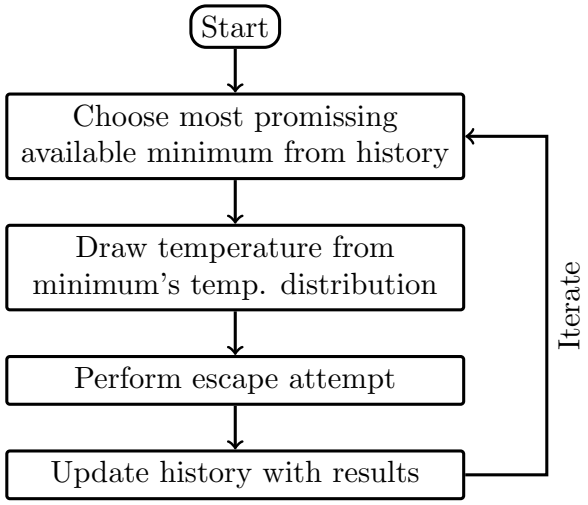


Figure 5.1: Flowchart of the Minima Crawling scheme.

5.1 Choosing a Promising Minimum

In order to choose a minimum as a starting configuration it has to be available. For this two conditions have to be met:

1. The minimum must be active. New minimums are always added as active, but they can get deactivated over the course of the optimization.

2. The minimum must not already be in use by too many other workers. The maximum number of worker that are allowed to simultaneously work on a minimum is a user-defined parameter, a typical value is three.

Among the available minima the most promising minimum is chosen based on a scoring function. The minimum with the lowest score is chosen. The score of a minimum is defined as the average over its list of *escape energies*. This list contains the energies of the last minima that were found when it was used as a starting configuration. Hence, the escape energies of a minimum are something like the track record of the minimum's recent performance. After all "nearby" minima are discovered, the performance of a minimum degrades. In order to account only for recent escapes, the list of escape energies is kept rather short. The length is a user-defined parameter, a typical value is three.

Once a minimum has been chosen, its number of active workers is increased by one. Limiting the maximum number of active workers per minimum prevents that all workers chose the same minimum and perform redundant work.

5.2 Drawing a Temperature

In the Minima Hopping scheme the temperature is increased when a previously found minimum is encountered again. This includes the case that the escape attempt fails and ends up at the original starting configuration. When a new minimum is discovered the temperature is decreased. This scheme tunes the temperature such that on average a new minimum is found during every second escape attempt.

The Minima Hopping scheme chooses temperatures with a 50% success probability without explicitly maintaining the corresponding probability distribution. For a parallelized scheme this probability distribution has to be maintained explicitly in order to allow for the different workers to combine their information. Therefore, in the Minima Crawling scheme every minimum has an associated temperature distribution $\rho(T)$. It gives for each temperature the probability for an escape attempt to succeed. The distribution is discretized in temperature steps T_i given by:

$$T_i = \beta^i \quad \text{with } \beta > 1\,. \tag{5.1}$$

The parameter β plays the same role as in the Minima Hopping scheme. There the temperature is increased by repeatedly multiplying with β, which leads to the same discretized exponential increase.

For choosing the temperature for an escape attempt, the distribution $\rho(T)$ is transformed in the following way:

$$\theta(T) = 1 - 2\,|\rho(T) - 0.5|\,. \tag{5.2}$$

The derived distribution $\theta(T)$ gives the probability that the success-rate at a given temperature is near 50%. The new temperature is then sampled from this distribution $\theta(T)$. The sampling is performed by drawing a temperature step i and a floating point number α between 0 and 1 from a random number generator. When $\theta(T_i) > \alpha$ is fulfilled, the temperature T_i is used for the escape attempt, otherwise new sets of random numbers are generated until the condition is fulfilled.

As initial temperature distribution ρ_0 a Fermi-like step-function centered around an initial temperature T_a is used:

$$\rho_0(T_i) = \frac{1}{1 + e^{\frac{a-i}{\tau}}}\,. \tag{5.3}$$

The slope of the step is controlled by the user-provided parameter τ, a typical value is $\tau = 5$. However, the importance of the initial temperature distribution is small, because it's only used for the first discovered minimum. The following new minima inherit the temperature distribution from their starting minimum.

5.3 Updating the Temperature Distribution

After an escape attempt with a certain temperature T_j was performed, the newly gained information has to be incorporated into the starting minimum's temperature distribution. For this the distribution is modified around T_j by adding or subtracting a small Gaussian:

$$\rho'(T_i) = \max\left[0,\ \min\left[1,\ \rho(T_i) \pm \epsilon \cdot e^{-\left(\frac{j-i}{\sigma}\right)^2}\right]\right] . \tag{5.4}$$

In (5.4) the "+" sign is used to increase the success probability if the attempt was successful, the "-" sign is used to decrease the success probability if the attempt failed. The protection with max() and min() ensures that $0 \leq \rho \leq 1$ remains satisfied. The width σ and the height ϵ of the Gaussian are user-parameters, typical values are $\sigma = 2$ and $\epsilon = 0.2$.

Figure 5.2 shows a temperature distribution $\rho(T)$ together with its derived distribution $\theta(T)$. The distribution was recorded only a few steps after the Minima Crawling scheme was launched. Therefore, it still shows the remains of the initial Fermi-distribution around 10 K. The bumps in the distribution are caused by the updates with Gaussians. Over the course of a Minima Crawling run, the step in the temperature distribution tends to become much steeper as the optimal temperature is narrowed down.

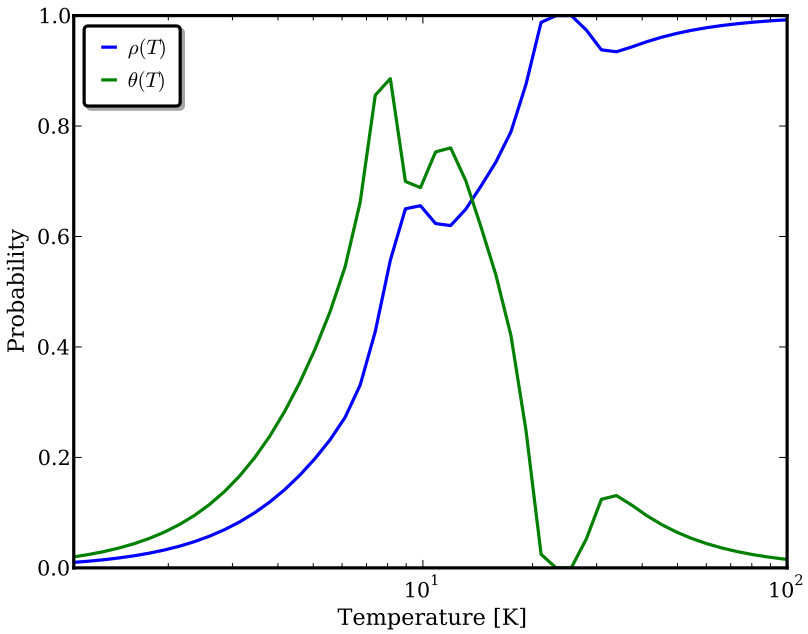


Figure 5.2: A temperature distribution $\rho(T)$ together with its derived distribution $\theta(T)$. The distribution was recorded from a Minima Crawling run shortly after the launch.

5.4 Adding a New Minima

When an escape attempt was successful and a new minimum is found, it is added to the history of encountered minima. The list of escape energies for the new minimum is initialized by setting all values to the minimum's own potential energy. The temperature distribution of the new minimum is initialized with the distribution of the starting minimum.

Furthermore, the potential energy of the newly encountered minimum is added to the list of escape energies of the starting minimum. The list has a fixed length and works as a *first in, first out queue*. Hence, the oldest entry gets removed in the process.

If the new minimum has a lower energy than the starting minimum, the starting minimum is disabled to prevent it from being sampled any further. This is equivalent to the Minima Hopping scheme, where a minimum with a lower energy is always accepted. Since a minimum is only disabled when a new minimum gets added, the number of active minima never decreases.

6 Implementation

In this chapter the implementation of the previously introduced methods in CP2K is presented. The code is split into two parts: At first a novel parallelization framework is implemented. Within this framework the actual optimization algorithms are implemented as plugins.

Global optimization algorithms typically operate via iterative exploration of the parameter space. New information is obtained by evaluating the optimization objective at promising points in the parameter space. Based on the new information the next evaluation points are chosen. This evaluation of the objective usually dominates the computational costs of a global optimization. In order to speedup an optimization, the objective can be evaluated at multiple points in parallel, because these are independent tasks.

In the context of geometry optimization of molecular systems, the optimization objective is the potential energy. In the field of computational chemistry the potential energy is very often calculated with so-called *self consistent field* methods [54]. These methods are iterative algorithms with a varying number of iteration steps. Hence, the computation time required to calculate the potential energy varies from one geometry to the next.

Many parallelization schemes are organized in a lock-step fashion, where all processors have to finish their work at the same time. In the case of a global optimization the tasks require different amounts of time. It is therefore not suitable for a lock-step scheme. Instead a master/worker scheme is more appropriate, in which the master process assigns tasks to the worker processes, the workers perform their assigned tasks and send their results back to the master asynchronously.

6.1 Swarm-Framework

The swarm-framework implements such a master/worker scheme in CP2K. The name *swarm* was chosen, because the workers collectively solve a common problem. This is similar to the swarm behavior observed with certain animals, e.g. ants and bees.

The architecture of the framework is based on the simple picture of a master having a dialog with its workers (see Figure 6.1). The master sends a command to a worker, the worker executes the command, replies with a report and awaits the next command. The entire process is bootstrapped by sending reports with the status "initial_hello" to the master.

The design goal of the swarm-framework was to make it as simple as possible to implement new optimization algorithms as plugins. The framework is clearly separated from the plugins and takes care of most of the technical and boilerplate work.

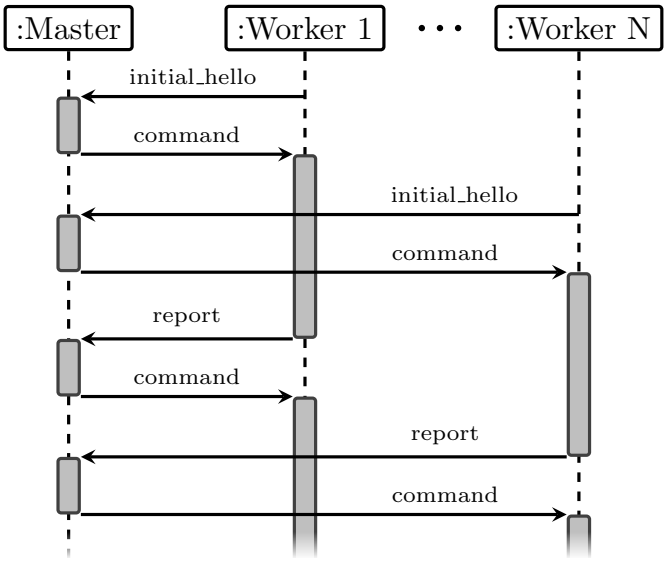


Figure 6.1: Sequence diagram illustrating the communication between the master and workers within the swarm-framework, including bootstrapping.

A major design decision was to make the main driver loops part of the swarm-framework. As a consequence the plugins are merely called by the framework when needed. On the workers the routine `execute` is called. As arguments it gets passed in a command and is expected to return a report. On the master the routine `steer` is called. It gets passed in a report from a worker and is expected to return a new command for the same worker. Commands and reports are stored as the `swarm_message` data-type. The entire scheme of the swarm-framework is shown in Figure 6.2. The green and red arrows represent the exchange of swarm-messages.

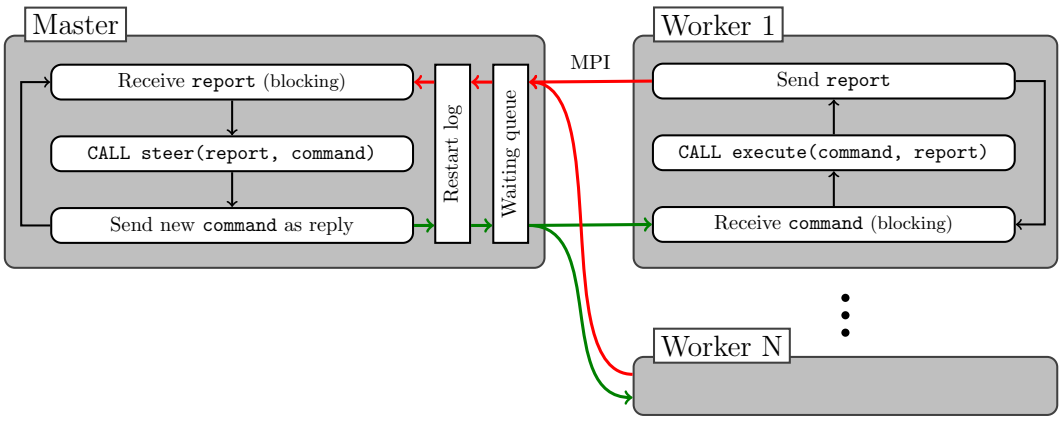


Figure 6.2: Architecture overview of the swarm-framework. The black arrows indicate control flow, the red arrows indicate the exchange of reports, the green arrows indicate the exchange of commands.

Swarm-Messages

A swarm-message is a data structure, which stores key/value pairs and is used for any communication between the master and the workers. The keys in a swarm-message are always strings, while the values can also be integer or floating point numbers or an array of numbers. The swarm-message is very easy to use. Entries are added with the routine `swarm_message_add` and retrieved with `swarm_message_get`. This programming convenience comes at some performance costs. In any case, the swarm-framework is not intended for communication dominated algorithms.

In high performance computing the *Message Passing Interface* (MPI) is the de facto standard API for network communication [57]. A group of communicating processors is represented by a *MPI communicator*. Within a communicator each processor is assigned a unique address, called a *MPI rank*. A processor may belong to multiple communicators.

By using the swarm-messages the plugin-programmer does not have to deal with the MPI at all. The swarm-framework transparently serializes and deserializes the message for transport over the network with MPI. While the master is always assigned a single process, each worker can consist of several MPI-processes. This allows for further parallelization, because many routines in CP2K, which the workers utilize, are MPI parallelized.

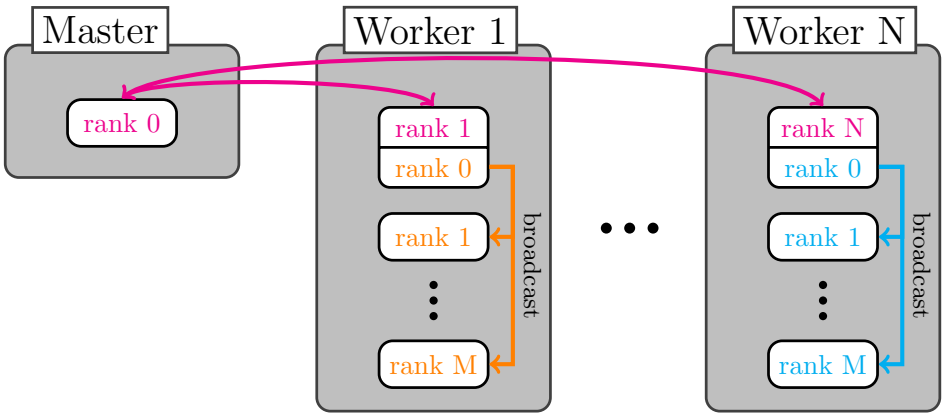


Figure 6.3: Communication scheme adopted by the swarm-framework. Arrows indicate MPI communication, color indicates the associated MPI communicator.

The communication scheme adopted by the swarm-framework is illustrated in Figure 6.3. The arrows indicate MPI communication, their color indicates the associated MPI communicator. The purple communicator contains the master process and the first process of each worker, i.e. the *foreman processes*. Additionally, there exists one communicator per worker, which contains all its processes. As a consequence, each foreman process belongs to two MPI communicators, in which it takes a different MPI rank.

In CP2K the convention was adopted that after a MPI parallelized operation has finished all processors have to be in the same state. For example, after performing a local geometry optimization all processes posses the final position and potential energy values. Hence, the

foreman process has all necessary information to assemble and send reports to the master. However, when receiving a command from the master, the foreman process has to broadcast this information to the other processes belonging to its worker to fulfill the convention.

Single Worker Mode

The swarm-framework was developed to coordinate a large number of workers. However, sometimes it is necessary to run with only a single worker. This was for example the case, when the serial performance of the Minima Hopping algorithm was benchmarked. In this situation a separate master process is not needed, running one anyways would be a waste of resources. Therefore, the swarm-framework has a special *single worker mode*, which requires only a single process to run. Instead of sending commands and reports over MPI they are just passed on locally between the master and the worker code. The code of the main driver loop for this mode is listed in Figure 6.4.

```
1  CALL swarm_master_init(master, n_workers=1, ... )
2  CALL swarm_worker_init(worker, worker_id=1, ... )
3  CALL swarm_message_add(report, "worker_id", 1)
4  CALL swarm_message_add(report, "status", "initial_hello")
5
6  DO WHILE(TRIM(command) /= "shutdown")
7     CALL swarm_master_steer(master, report, cmd)
8     CALL swarm_message_free(report)
9     CALL swarm_worker_execute(worker, cmd, report)
10    CALL swarm_message_get(cmd, "command", command)
11    CALL swarm_message_free(cmd)
12 END DO
13
14 CALL swarm_message_free(report)
15 CALL swarm_worker_finalize(worker)
16 CALL swarm_master_finalize(master)
```

Figure 6.4: Code listing of the main driver loop for the single worker mode.

Generic Restart Mechanism

High performance computations are often interrupted by time limits or failures. Hence, it is vital that a computation can be restarted close to the point were it was left off. The implementation of a restart mechanism requires a serialization of the program's internal state and a way to restore it later. The set of variables that make up the internal state depends on the algorithm employed. Therefore, usually for each algorithm a special serialize and restore routine has to be written.

The swarm-framework has a generic restart mechanism build in. It works by recording all communication between master and workers. For a restart the communication is simply replayed to the master. When the end of the communication log is reached, the last command appointed to each worker is send out and the computation continues in a normal fashion.

The generic restart mechanism is based on three assumptions:

1. The master's `steer`-routine is deterministic. During a restart, the master is presented with the previously recorded reports and it has to reply by issuing exactly the same commands again. The commands are compared with the recorded ones, when they don't match the restart is aborted.

2. The master's `steer`-routine executes quickly. During a restart, the master's state is restored by stepping through the entire computation in an accelerated fashion. The time required to perform a restart basically depends on the execution speed of the `steer`-routine, because the worker's reports are on file and instantly available. If the `steer`-routine performs too heavy computation a restart will consume an unreasonable amount of time. In this case, work should be migrated from the master to a worker by launching additional tasks.

3. The workers are stateless. During a restart, the workers wait idle for their first command. When the entire communication log has been replayed to the master, the workers are sent only the latest command appointed to them. Based on this single command they must resume their operation as if they were never interrupted. Therefore, the workers must not have any internal state. This restriction could be mitigated somewhat by allowing for a few stateful commands to be send in between stateless commands. Restarts would then always continue after stateless commands. However, this feature was not yet needed and was therefore not implemented as part of this thesis.

Waiting Queue

After a worker has send its report it awaits a new command in reply. However, sometimes the master does not have an immediate task for the worker. This occures when the master has to receive reports from other workers as well before generating new tasks. In this situation a few workers have to idle for a short time. The simplest way to implement this would be to command the worker to sleep for a certain period of time, they would then send back a "wake-up" report. However, choosing this time period is difficult: If the time is chosen too short, the master might get overwhelmed with "wake-up" reports. If the time is chosen too long the workers notice too late that new tasks are available and computer time would be wasted.

The swarm-framework offers a solution to this problem, which allows to retain the simple architecture without losing performance: Whenever the master has no immediate new task available it just returns the command "wait". The swarm-framework will filter this command and it is not passed on to the worker. Instead the worker's id is added to a list of waiting workers. Since each incoming report provides the master with new information, it might trigger the generation of new tasks. Therefore, whenever the master receives a report it is afterward given the opportunity to also send commands to the waiting workers. For this purpose a report with the status "wait_done" is emitted for each waiting worker.

From the master's perspective the workers are actually going to sleep, but they just happen to always wake up right after a new report was received. The workers do not notice any

of this. They will just remain in a blocking call to `mpi_recv` until the master sends out a real command. In order to not clutter up the communication log, continued waiting is not recorded, i.e. "wait_done"-reports replied by with another "wait"-command. Of course, the master should never ask all of its workers to wait at the same time, but this would make no sense anyways.

6.2 Minima Hopping and Crawling

Based on the swarm-framework the Minima Hopping and the Minima Crawling schemes were implemented. The basic operation of both schemes is to perform an escape attempt. Each escape attempt requires several force evaluations, which constitutes the main computational costs. Therefore, this task is off-loaded to the workers and implemented in their `execute` routine. Both optimization schemes use the same implementation for their workers.

A worker receives for each escape attempt a set of coordinates and a temperature. The worker then performs all the steps as described in Chapter 4. At the end of the local optimization the worker sends back a report, which contains the final configuration and its potential energy.

Each worker has its own trajectory file. To this file the intermediate configurations of the MD and the local optimization are written. The frames in a trajectory file are numbered consecutively. However, CP2K does not keep track of the last written frame number. Hence, the current frame number has to be passed explicitly when calling the MD simulation and the local optimization. In order for the generic restart mechanism to work properly, the worker has to be state-less. As a consequence, the frame number can not be stored on the worker in-between the execution of commands. It is therefore send back to the master as part of the report. The master will return the frame number to the worker as part of the command for the next escape attempt.

Each optimization scheme implements its own `steer` routine. However, they both utilize a common history implementation. The history provides the functionality for storing and recognizing visited minima. It calculates descriptors of the minima and compares them with respect to a configurable threshold. The clean separation of the history from the actual optimization algorithm allows for easy code changes. For example, in the future an improved descriptor might allow to loosen the convergence criteria on the local optimization, because minima are recognized more reliably.

The remaining code for the implementation of the actual optimization algorithms is very compact. For example, the Minima Hopping scheme requires less than 300 lines of Fortran code. This shows, that with the swarm framework in place it is now very easy to add new optimization schemes to CP2K.

Besides implementing the main functionality for the global optimization schemes, also some other small changes were made: CP2K is highly optimized for performing quantum chemical calculations. However, using the cheap Lennard-Jones potential drives the code into a regime where it's not well optimized. For example, a two-fold speedup could be achieved by improving the way CP2K does its log-handling. This little anecdote should serve as an example for the many small technical difficulties that were encountered with CP2K's internal facilities.

They include program start-up, log-handling, error-handling, profiling, message passing, input parsing, random number generation, and regression testing. For most of these aspects there exists no documentation and one has to revert to the code itself for information.

7 Benchmarks

In this chapter the performance of the implemented global optimization schemes is benchmarked and compared. For all benchmarks the Lennard-Jones 38 cluster is used. As explained in Chapter 2, it is a widely recognized benchmarking system for global optimization algorithms. It has the advantage of having an energy function that is very cheap to evaluate, which allows to run an extensive number of tests. The global minimum of the LJ_{38} cluster is well known, but it is considered hard to find, because it's located in a narrow side funnel, which is entropically unfavored.

The optimization runs are always started from the same configuration. In order to have an unbiased starting point the particles are laid out on a regular grid with a spacing of 1.5 Å (see Figure 7.1), where 1 Å = 0.1 nm denotes the ångström length unit.

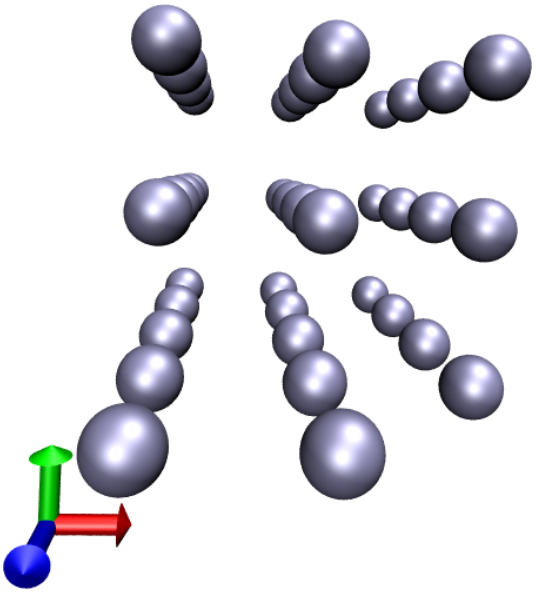


Figure 7.1: Starting configuration of the LJ_{38} cluster used for benchmarks.

In order to obtain energy values with a magnitude that is typical for chemistry simulations the Lennard-Jones potential is parametrized with $\sigma = 1.0$ Å and $\epsilon = 0.001 E_H$, where $E_H \approx 27$ eV denotes the Hartree energy unit. Internally, the LJ potential is approximated with a spline using 2038 supporting points, which provides an accuracy below $10^{-10} E_H$. For particles which are more than 25 Å apart the LJ potential is cutoff and considered to be zero. The particles are assigned the mass of one atomic mass unit.

In the Minima Hopping scheme the initial temperature is $T_0 = 10\,\mathrm{K}$. It is increased or decreased by the factor $\beta = 1.1$. The initial acceptance energy is $E^0_{\mathrm{accept}} = 0.005\,E_\mathrm{H}$. It is increased or decreased by the factor $\alpha = 1.02$.

In the Minima Crawling scheme the initial temperature distribution uses a step-function centered around $T_0 = 10\,\mathrm{K}$ with a slope-parameter of $\tau = 5.0$. The distribution is updated by adding or subtracting Gaussians with a width of $\sigma = 2.0$ and a height of $\epsilon = 0.2$. For each minimum a list of escape energies records the last three successful escapes. The number of active workers per minimum is limited to three.

The initial velocities of the MD are softened by performing 20 softening iterations. The initial test point for the softening is chosen at a distance of $d = 0.01\,\mathrm{a_0}$, where $a_0 \approx 0.5\,\text{Å}$ denotes the Bohr radius. In each softening iteration the test-point is updated using a mixing-factor of $\alpha = 0.5$.

Two minima are considered equal if their energies are closer than $5.0 \cdot 10^{-5}\,E_\mathrm{H}$ and their descriptors are closer than 0.01. When a run discovers a configuration, which has a potential energy lower than $-0.1739\,E_\mathrm{H}$, it is considered as the global minimum of the LJ_{38} cluster and the run is terminated.

For the local optimization the conjugate gradient algorithm is used. Any other settings, e.g. regarding the MD or the local optimization, remained at the default values of CP2K.

7.1 Serial Minima Hopping

At the beginning of each escape attempt the initial velocities are drawn from a normal distribution using a random number generator. This makes the Minima Hopping algorithm very stochastic in nature. In fact, depending on the seed of the random number generator a run might finish within a few steps or it might require several thousand steps to finish. Therefore, only a statistical assessment of the performance is possible. For this the algorithm has to be run several times with different seeds of the random number generator.

Figure 7.2 shows the performance of the serial Minima Hopping algorithm based on 1000 independent runs. For each run the number of required force evaluations and the wall time are recorded. Then the *Cumulative Distribution Function* (CDF) for each quantity is calculated. It shows the probability of a single run to find the global minimum after a certain time or number of force evaluations. For example, after 3.2 M force evaluations there is a 90% probability that a run has already found the global minimum. The two distributions look very similar, because in the serial case the number of force evaluations is proportional to the runtime. The CDFs allows to quickly estimate the computer time required to find the global minimum.

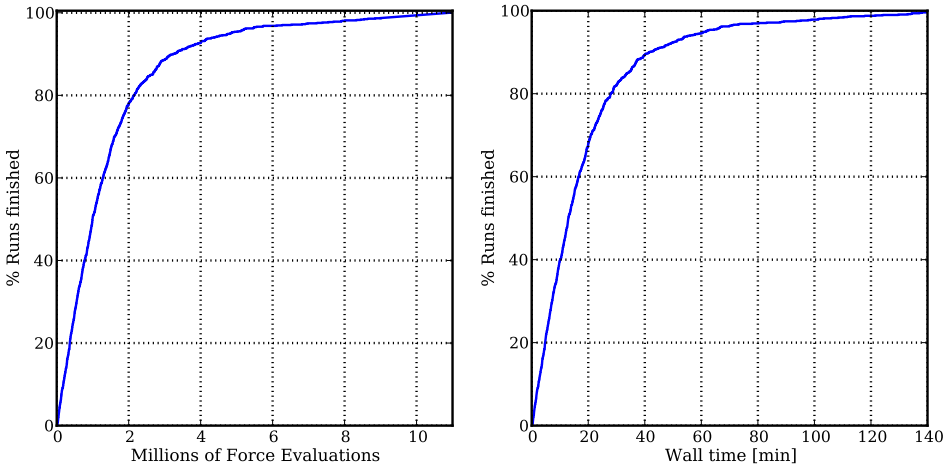


Figure 7.2: CDFs of the wall time and the number of force evaluations required by the serial Minima Hopping algorithm to find the global minimum. The plots are based on 1000 independent runs.

7.2 Velocity Softening

The velocity softening method adjusts the initial velocities of an escape attempt to make them more favorable to encounter a low barrier. The method was described in detail in Section 4.2. Velocity softening is an iterative method with a fixed number of steps. The number of steps is a delicate parameter to choose: If too few softening steps are performed the initial random velocities remain basically unchanged. If too many softening steps are performed the initial velocities converge onto the lowest eigenmode and become deterministic. Since the Minima Hopping method relies on a certain amount of randomness, a good balance has to be found. Furthermore, each softening step comes at the cost of an additional force evaluation.

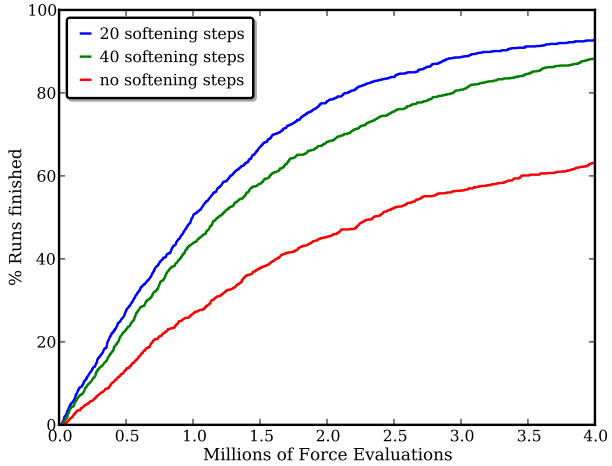


Figure 7.3: Comparison of the effect of different number of softening steps on the optimization speed. Each plot is based on 1000 independent runs.

Schönborn et al. [40] reported that 20 softening steps yield a good result for the Lennard-Jones 38 cluster. In order to verify this, simulations are run with 0, 20, and 40 softening steps. Figure 7.3 compares the CDF obtained, and it confirms that 20 softening steps yield good results.

7.3 Embarrassingly Parallel Minima Hopping

The number of force evaluations required by the Minima Hopping method depends strongly on the chosen seed for the random number generator. An obvious way to parallelize the method is therefore the launch of multiple independent workers with different seeds. All workers are terminated as soon as one of them finds the global minimum. A scheme like this, in which no communication between the parallel tasks is required, is called *embarrassingly parallel*.

When tasks are independent, the performance of the parallel run can be derived from the performance of the serial run: Let $F(k)$ denote the CDF used to describe the serial performance in terms of force evaluations as shown on the left side of Figure 7.2. It gives the probability of a serial run to finish after a given number of force evaluations k. The probability of a serial run to **not** finish after k force evaluations is therefore given by $1 - F(k)$. The parallel run will continue to run until one of its n workers has found the global minimum. Hence, the probability for the parallel run to **not** finish after a total of k force evaluations is given by $[1 - F(k/n)]^n$. Finally, the probability of a parallel run with n workers to finish after a total of k force evaluations is given by:

$$F_n(k) = 1 - \left[1 - F\left(\frac{k}{n}\right)\right]^n . \tag{7.1}$$

When the number of workers n is increased, two opposing effects occur: On one hand, the term $\left[1 - F\left(\frac{k}{n}\right)\right]^n$ will quickly decay to zero when $F(k/n)$ becomes larger than zero. This reflects that with more workers there is a higher chance that one of them will finish quickly. On the other hand, the CDF of the serial runs enters in a stretched form as the term $F(k/n)$. This reflects that until one worker finishes all the other workers perform force evaluations as well. As a result the shape of F_n is determined by the shape of $F(k)$ for very small values of k.

In order to obtain a thorough sampling of the onset of $F(k)$ a set of 10.000 serial runs is performed. Runs that have not finished after 1000 escape attempts are aborted. This is roughly equivalent to 0.6 M force evaluations. From this the CDF for $k \leq 5.5 \cdot 10^5$ is calculated as shown on the left side of Figure 7.4. This CDF is then fitted with a polynomial function of degree 10. Based on this fit the CDFs for parallel runs are calculated using (7.1) and shown on the right side of Figure 7.4. The calculated CDFs for the parallel runs show clearly that the number of required force evaluations increases with an increasing number of workers.

The same analysis is performed for the wall time and shown in Figure 7.5. The CDF of the wall time for the parallel case is calculated using the following formula:

$$F_n(t) = 1 - [1 - F(t)]^n . \tag{7.2}$$

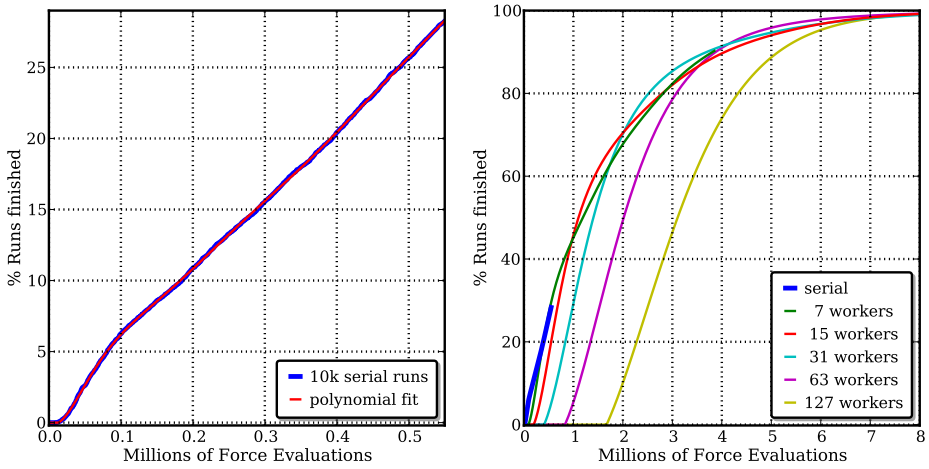


Figure 7.4: left: Thorough sampling of the onset of the CDF of the number of force evaluations for the serial Minima Hopping using 10.000 independent runs. right: Comparison of calculated CDFs for embarrassingly parallel runs with different number of workers.

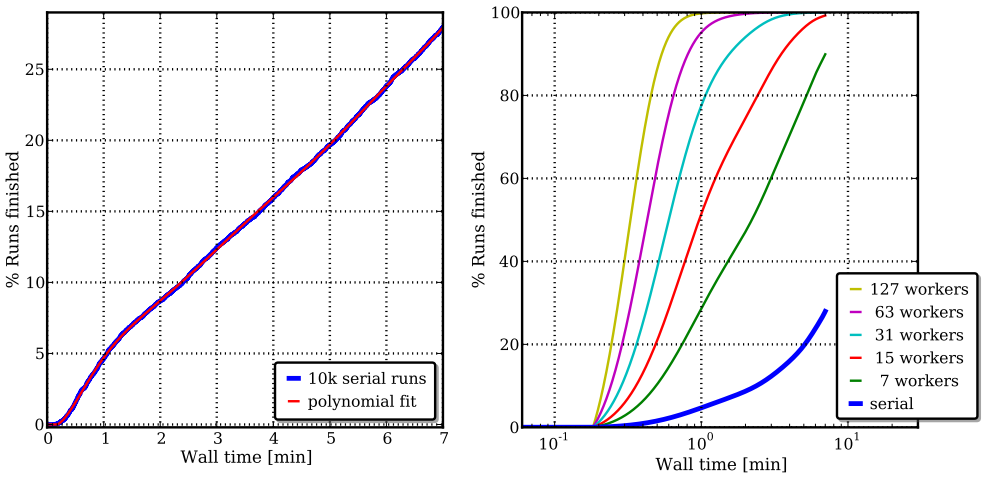


Figure 7.5: left: Thorough sampling of the onset of the CDF of the wall time for the serial Minima Hopping using 10.000 independent runs. right: Comparison of calculated CDFs for embarrassingly parallel runs with different number of workers.

In comparison to (7.1), the CDF of the serial run $F(t)$ enters into $F_n(t)$ unstreched. As a result $F_n(t)$ converges for an increasing number of workers n towards a step-function around $t = 0.2\,\text{min}$. This is the time the fastest of the 10.000 serial runs took to finish. In order to accommodate for the large range of different wall times, log-scaling is used for these axes. Some of the curves in Figure 7.4 and 7.5 do not reach 100%, because the sampling of $F(k)$ and $F(t)$ was limited to only 1000 escape attempts.

7.4 Shared History Minima Hopping

Schönborn et al. suggest to parallelize the Minima Hopping scheme by letting the workers share their histories of visited minima. The idea is that due to the feedback mechanism and the common history an overlap of search area will be penalized [40].

The possibility to let the Minima Hopping workers share a common history is also implemented in CP2K. In order to investigate the performance of this scheme its CDFs for different number of workers are measured. Each CDF is based on a set of 500 runs. The number of workers is chosen as $2^m - 1$. Together with the master process this results in powers of two for the number of processors, which is favorable for batch processing. The CDFs obtained from the runs with history sharing are shown in Figure 7.6. The distributions show that the performance of the history sharing scheme is very similar to the performance of the embarrassing parallelization. For example, both schemes require between 2 and 3 M force evalutations to reach 80% finnished runs with 7-31 workers. With 127 workers both schemes require slightly more than 4 M force evalutations to reach 80%.

7.5 Minima Crawling

The Minima Crawling method is a novel scheme, which was developed as part of this thesis. Its aim is to achieve a better parallel performance than the Minima Hopping scheme, while retaining its successful key ideas. The performance of the Minima Crawling scheme is measured in the same way as the Minima Hopping scheme. Again, sets of 500 runs using 7, 15, 31, 63, and 127 workers are used.

The obtained CDFs are shown in Figure 7.7. Compared to the Minima Hopping scheme, the distributions of the Minima Crawling scheme show better performance. Especially, the tails of the CDFs are shorter, which can also be seen from the fact that basically all runs finish after 6 M force evaluations.

For smaller numbers of workers the Minima Crawling scheme makes slower progress in the beginning of the optimization than the Minima Hopping scheme. A reason for this might be that the parameters used for the benchmarks were tuned for a large number of workers. For example, using a maximum of three active worker per minimum might be too much if the optimization is run with only seven workers.

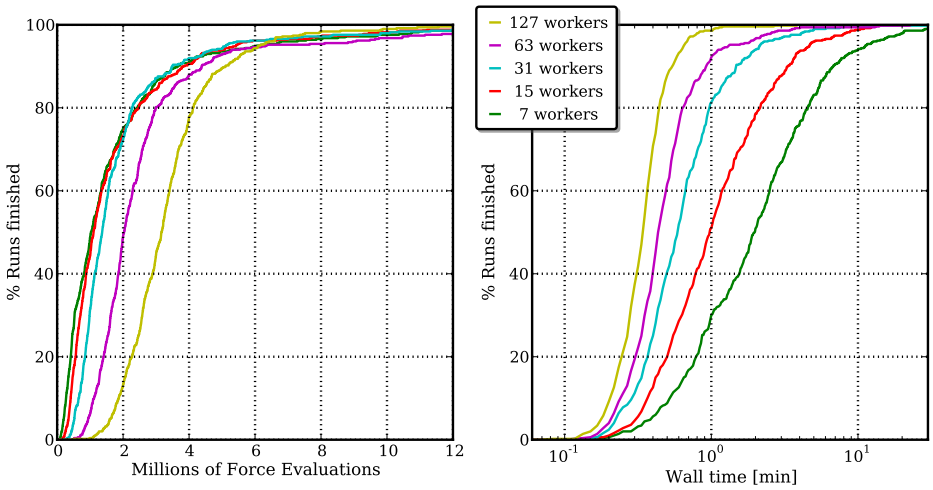


Figure 7.6: CDF of the force evaluations and wall time required to find the global minimum using the Minima Hopping scheme with history sharing and different numbers of workers. Each curve is based on 500 independent runs.

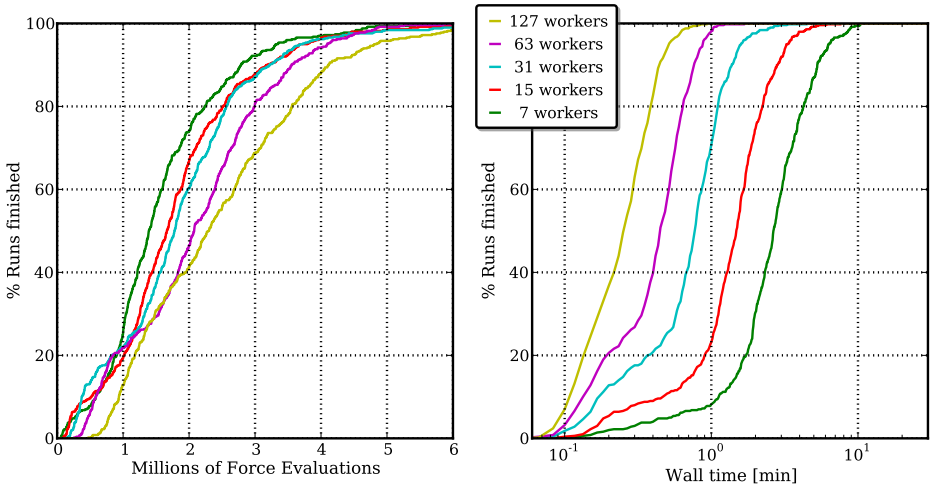


Figure 7.7: CDF of the force evaluations and wall time required to find the global minimum using the Minima Crawling scheme and different numbers of workers. Each curve is based on 500 independent runs.

7.6 Comparison of Parallel Performance

In order to obtain a quantitative comparison of the three parallelization schemes the 90% quartiles of their CDFs are calculated and shown in Figure 7.8. For the embarrassing parallelization the analytic model allows to create smooth curves, while for the other two schemes the data points for 7, 15, 31, 63, and 127 workers are shown.

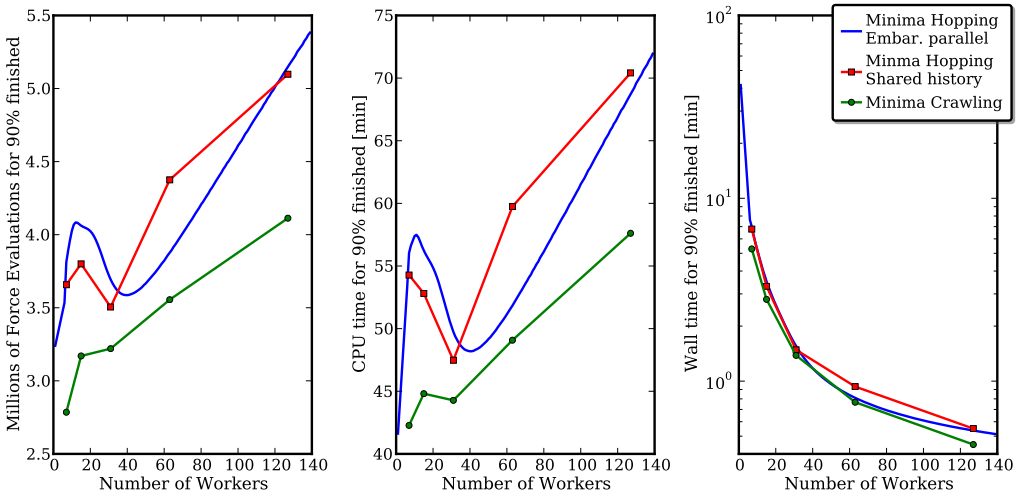


Figure 7.8: Comparison of performance, costs and wall time for the different parallelization methods based on the 90% quartiles of the CDFs.

The left plot of Figure 7.8 shows the number of force evaluations needed to reach the 90% quartile. The serial Minima Hopping run requires 3.2 M force evaluations to reach this quartile. For the embarrassing parallelization of Minima Hopping the curve quickly climbs to 3.8 M evaluations for 7 worker and 4.1 M evaluations for 13 workers. Afterward, it drops back down to 3.6 M evaluations for 40 workers. This local minimum of the curve originates from the slightly steeper increase in the CDF of the serial run below 5%, as shown in Figure 7.4. As the number of workers increases this early onset of the CDF becomes predominant and the optimization progresses quicker. For more than 40 workers the performance improvements are diminishing. The curve increases linearly with a slop of 0.02 M force evaluations per worker. This means that the fastest worker finds the global minimum after 0.02 M force evaluations and the additional workers waste force evaluations while they "wait" for this.

The performance of the Minima Hopping scheme with shared history is very similar to its embarrassing parallelization. For most data points the history sharing has slightly better performance than the independent workers, but for 63 workers it requires 0.6 M evaluations more. Therefore, the history sharing scheme seems to give no significant advantage over the embarrassing parallelization.

The Minima Crawling scheme shows a consistently better performance than the Minima Hopping scheme. It requires only 2.8 M force evaluations with 7 workers to reach the 90% quartile. This is one million less than the Minima Hopping scheme requires. As the number of workers increases the Minima Crawling also requires more force evaluations. For 127 workers

the scheme needs 4.1 M evaluations, while the embarrassing parallelization of Minima Hopping requires 5.2 M. The curve's increase for Minima Crawling is not as steep as for the Minima Hopping scheme, which suggests that its performance will remain superior even for higher numbers of workers.

The right plot of Figure 7.8 shows the wall time required to reach the 90% quartile. It shows that the parallelization with only 7 workers already leads to a significant reduction of the wall time from 41.6 minutes for the serial run down to 5.3 minutes for the Minima Crawling scheme. The embarrassingly parallel Minima Hopping with 7 workers requires 7 minutes to reach the quartile. As the number of workers increases the wall time decreases further, but the differences between the schemes become negligible. With 127 workers the wall times of all schemes has basically converged onto 30 seconds, which is again the time the quickest worker needs to finish.

The middle plot of Figure 7.8 shows the CPU time required to reach the 90% quartile. CPU time is the product of the wall time and the number of utilized processors. It is a realistic measure for the actual cost of the calculation. The serial run requires 41.6 minutes of CPU time to reach the quartile using one processor. The Minima Crawling scheme with 7 worker uses 8 processors and reaches the quartile after 42.3 CPU minutes. Hence, the Minima Crawling with 7 workers offers an almost eight-fold speedup in time to solution, at basically no additional cost.

It should be stressed that all the previous observations were made in the context of the Lennard-Jones 38 cluster. For a more complex system it might take considerably more steps to finish a run, even for the quick workers. The performance of the embarrassing parallelization is very sensitive to the onset of the serial CDF. It can therefore be anticipated that different molecular systems might favor different strategies. An in-depth study of this is beyond the scope of this thesis.

8 Summary

In this thesis the CP2K software package was extended with the functionality for performing global geometry optimization. Most of this functionality is provided by the novel swarm-framework. The framework follows a master/worker software architecture, which is very suitable for global optimization algorithms.

Based on the swarm-framework the Minima Hopping method, which was introduced by Stefan Goedecker, was implemented [6]. Two parallelization schemes for Minima Hopping were implemented: An embarrassing parallelization using multiple independent workers and a scheme based on a shared history as proposed by Schönborn et al. [40].

Furthermore, a new optimization scheme, called Minima Crawling, was developed and implemented as part of this thesis. The aim was to to make better use of the collective information that the central master process obtains from all its workers, while retaining the successful key ideas of Minima Hopping.

Afterward, the different methods were compared using the established benchmark system of a Lennard-Jones cluster with 38 particles. The benchmarks showed that already a small number of workers leads to a significant reduction of the run time for all schemes. Furthermore, the Minima Crawling indeed delivers better performance than the parallelized Minima Hopping scheme. With 7 workers the Minima Crawling offers an almost eight-fold speedup in time to solution at virtually no additional cost, compared to the serial Minima Hopping algorithm.

The swarm-framework greatly simplifies the implementation of global optimization algorithms in CP2K. For example, the implementation of the Minima Hopping scheme consists of less than 300 lines of Fortran code. The hope is that the swarm-framework lays the foundation for the development and implementation of more and improved optimization algorithms in the future.

A major part of the work for this thesis was devoted to programming. While a simple stand-alone implementation would presumably have taken only a few weeks to finish, the integration into CP2K has proven to be far more challenging. Nevertheless, the code was accepted into the official CP2K svn-repository on the 26^{th} of November 2013 as revision 13355. It is now publicly available at *sourceforge.net*. Soon the new features will also become available in popular Linux distributions such as *Debian* and *Fedora*, when the next release version of CP2K is packaged.

Bibliography

[1] J. Ngo, J. Marks and M. Karplus, *Computational complexity, protein structure prediction, and the levinthal paradox*, in J. Merz, KennethM. and S. Grand, eds., *The Protein Folding Problem and Tertiary Structure Prediction*, 433–506, Birkhäuser Boston (1994), doi:10.1007/978-1-4684-6831-1˙14.

[2] L. T. Wille and J. Vennik, *Computational complexity of the ground-state determination of atomic clusters*, J. Phys. A **18**(8) L419 (1985), doi:10.1088/0305-4470/18/8/003.

[3] H. Sutter, *The free lunch is over: A fundamental turn toward concurrency in software*, Dr. Dobbs. J. **30**(3) (2005), URL http://www.gotw.ca/publications/concurrency-ddj.htm.

[4] H. Meuer, E. Strohmaier, H. Simon and J. Dongarra, *Top500 list* (2013), URL http://www.top500.org/list/2013/11/.

[5] J. VandeVondele, M. Krack, F. Mohamed, M. Parrinello, T. Chassaing and J. Hutter, *Quickstep: Fast and accurate density functional calculations using a mixed gaussian and plane waves approach*, Comput. Phys. Commun. **167**(2) 103 – 128 (2005), doi:10.1016/j.cpc.2004.12.014.

[6] S. Goedecker, *Minima hopping: An efficient search method for the global minimum of the potential energy surface of complex molecular systems*, J. Chem. Phys. **120**(21) 9911–9917 (2004), doi:10.1063/1.1724816.

[7] J. E. Jones, *On the determination of molecular fields. ii. from the equation of state of a gas*, Proc. R. Soc. London, Ser. A **106**(738) 463–477 (1924), doi:10.1098/rspa.1924.0082.

[8] I. E. Dzyaloshinskii, E. M. Lifshitz and L. P. Pitaevskii, *General theory of van der waals' forces*, Sov. Phys. Usp. **4**(2) 153 (1961), doi:10.1070/PU1961v004n02ABEH003330.

[9] W. Pauli, *Über den zusammenhang des abschlusses der elektronengruppen im atom mit der komplexstruktur der spektren*, Z. Phys. **31**(1) 765–783 (1925), doi:10.1007/BF02980631.

[10] R. A. Buckingham, *The classical equation of state of gaseous helium, neon and argon*, Proc. R. Soc. London, Ser. A **168**(933) 264–283 (1938), doi:10.1098/rspa.1938.0173.

[11] O. M. Becker and M. Karplus, *The topology of multidimensional potential energy surfaces: Theory and application to peptide structure and kinetics*, J. Chem. Phys. **106**(4) 1495–1517 (1997), doi:10.1063/1.473299.

[12] J. D. Bryngelson, J. N. Onuchic, N. D. Socci and P. G. Wolynes, *Funnels, pathways, and the energy landscape of protein folding: A synthesis*, Proteins **21**(3) 167–195 (1995), doi:10.1002/prot.340210302.

[13] S. Sastry, P. G. Debenedetti and F. H. Stillinger, *Signatures of distinct dynamical regimes in the energy landscape of a glass-forming liquid*, Nature **393**(6685) 554–557 (1998), doi:10.1038/31189.

[14] D. J. Wales, M. A. Miller and T. R. Walsh, *Archetypal energy landscapes*, Nature **394**(6695) 758–760 (1998), doi:10.1038/29487.

[15] D. J. Wales and J. P. K. Doye, *Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms*, J. Phys. Chem. A **101**(28) 5111–5116 (1997), doi:10.1021/jp970984n.

[16] Xiang, Jiang, Cai and Shao, *An efficient method based on lattice construction and the genetic algorithm for optimization of large lennard-jones clusters*, J. Phys. Chem. A **108**(16) 3586–3592 (2004), doi:10.1021/jp037780t.

[17] Xiang, Cheng, Cai and Shao, *Structural distribution of lennard-jones clusters containing 562 to 1000 atoms*, J. Phys. Chem. A **108**(44) 9516–9520 (2004), doi:10.1021/jp047807o.

[18] J. P. K. Doye, M. A. Miller and D. J. Wales, *The double-funnel energy landscape of the 38-atom lennard-jones cluster*, J. Chem. Phys. **110**(14) 6896–6906 (1999), doi:10.1063/1.478595.

[19] J. Doye, *Lennard-jones clusters*, URL `http://doye.chem.ox.ac.uk/research/forest/LJ.html`.

[20] J. P. K. Doye and D. J. Wales, *Thermodynamics of global optimization*, Phys. Rev. Lett. **80** 1357–1360 (1998), doi:10.1103/PhysRevLett.80.1357.

[21] S. Goedecker, W. Hellmann and T. Lenosky, *Global minimum determination of the born-oppenheimer surface within density functional theory*, Phys. Rev. Lett. **95** 055501 (2005), doi:10.1103/PhysRevLett.95.055501.

[22] K. Bao, S. Goedecker, K. Koga, F. Lançon and A. Neelov, *Structure of large gold clusters obtained by global optimization using the minima hopping method*, Phys. Rev. B **79** 041405 (2009), doi:10.1103/PhysRevB.79.041405.

[23] M. Amsler and S. Goedecker, *Crystal structure prediction using the minima hopping method*, J. Chem. Phys. **133**(22) 224104 (2010), doi:10.1063/1.3512900.

[24] S. Roy, S. Goedecker, M. J. Field and E. Penev, *A minima hopping study of all-atom protein folding and structure prediction*, J. Phys. Chem. B **113**(20) 7315–7321 (2009), doi:10.1021/jp8106793.

[25] P. S. P. Salomon and R. Frost, *Facts, Conjectures and Improvements for Simulated Annealing*, SIAM, Philaelphia (2002).

[26] B. A. Berg and T. Neuhaus, *Multicanonical algorithms for first order phase transitions*, Phys. Lett. B **267**(2) 249 – 253 (1991), doi:10.1016/0370-2693(91)91256-U.

[27] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, *Equation of state calculations by fast computing machines*, J. Chem. Phys. **21**(6) 1087–1092 (1953), doi:10.1063/1.1699114.

[28] H. Grubmüller, *Predicting slow structural transitions in macromolecular systems: Conformational flooding*, Phys. Rev. E **52** 2893–2906 (1995), doi:10.1103/PhysRevE.52.2893.

[29] A. F. Voter, *A method for accelerating the molecular dynamics simulation of infrequent events*, J. Chem. Phys. **106**(11) 4665–4677 (1997), doi:10.1063/1.473503.

[30] S. Stepanenko and B. Engels, *Gradient tabu search*, J. Comp. Chem. **28**(2) 601–611 (2007), doi:10.1002/jcc.20564.

[31] A. R. Oganov and C. W. Glass, *Crystal structure prediction using ab initio evolutionary techniques: Principles and applications*, J. Chem. Phys. **124**(24) 244704 (2006), doi:10.1063/1.2210932.

[32] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., New York, NY, USA (1989).

[33] C. Wehmeyer, G. F. von Rudorff, S. Wolf, G. Kabbe, D. Schaerf, T. D. Kuehne and D. Sebastiani, *Foraging on the potential energy surface: A swarm intelligence-based optimizer for molecular geometry*, J. Chem. Phys. **137**(19) (2012), doi:10.1063/1.4766821.

[34] S. Heiles and R. L. Johnston, *Global optimization of clusters using electronic structure methods*, Int. J. Quantum Chem. **113**(18) 2091–2109 (2013), doi:10.1002/qua.24462.

[35] S. Goedecker, *source code* (2013), personal communication.

[36] R. P. Bell, *The theory of reactions involving proton transfers*, Proc. R. Soc. London, Ser. A **154**(882) 414–429 (1936), doi:10.1098/rspa.1936.0060.

[37] M. G. Evans and M. Polanyi, *Inertia and driving force of chemical reactions*, Trans. Faraday Soc. **34** 11–24 (1938), doi:10.1039/TF9383400011.

[38] S. Roy, S. Goedecker and V. Hellmann, *Bell-evans-polanyi principle for molecular dynamics trajectories and its implications for global optimization*, Phys. Rev. E **77** 056707 (2008), doi:10.1103/PhysRevE.77.056707.

[39] G. Henkelman and H. Jónsson, *A dimer method for finding saddle points on high dimensional potential surfaces using only first derivatives*, J. Chem. Phys. **111**(15) 7010–7022 (1999), doi:10.1063/1.480097.

[40] S. E. Schönborn, S. Goedecker, S. Roy and A. R. Oganov, *The performance of minima hopping and evolutionary algorithms for cluster structure prediction*, J. Chem. Phys. **130**(14) 144108 (2009), doi:10.1063/1.3097197.

[41] W. C. Swope, H. C. Andersen, P. H. Berens and K. R. Wilson, *A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters*, J. Chem. Phys. **76**(1) 637–649 (1982), doi:10.1063/1.442716.

[42] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Clarendon Press, New York, NY, USA (1989).

[43] M. E. Tuckerman, *Statistical Mechanics: Theory and Molecular Simulation*, Oxford University Press (2010).

[44] J. Kiefer, *Sequential minimax search for a maximum*, Proc. Amer. Math. Soc. **4** 502–506 (1953), doi:10.1090/S0002-9939-1953-0055639-3.

[45] R. Fletcher and C. M. Reeves, *Function minimization by conjugate gradients*, Comput. J. **7**(2) 149–154 (1964), doi:10.1093/comjnl/7.2.149.

[46] P. Wolfe, *Convergence conditions for ascent methods*, SIAM Rev. **11**(2) 226–235 (1969), doi:10.1137/1011036.

[47] P. Wolfe, *Convergence conditions for ascent methods. ii: Some corrections*, SIAM Rev. **13**(2) 185–188 (1971), doi:10.1137/1013035.

[48] C. G. BROYDEN, *The convergence of a class of double-rank minimization algorithms 1. general considerations*, IMA J. Appl. Math. **6**(1) 76–90 (1970), doi:10.1093/imamat/6.1.76.

[49] R. Fletcher, *A new approach to variable metric algorithms*, Comput. J. **13**(3) 317–322 (1970), doi:10.1093/comjnl/13.3.317.

[50] D. Goldfarb, *A family of variable-metric methods derived by variational means*, Math. Comp. **24** 23–26 (1970), doi:10.1090/S0025-5718-1970-0258249-6.

[51] D. F. Shanno, *Conditioning of quasi-Newton methods for function minimization*, Math. Comp. **24** 647–656 (1970), doi:10.1090/S0025-5718-1970-0274029-X.

[52] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, 2nd ed. (2006).

[53] P. Hohenberg and W. Kohn, *Inhomogeneous electron gas*, Phys. Rev. **136** B864–B871 (1964), doi:10.1103/PhysRev.136.B864.

[54] W. Kohn and L. J. Sham, *Self-consistent equations including exchange and correlation effects*, Phys. Rev. **140** A1133–A1138 (1965), doi:10.1103/PhysRev.140.A1133.

[55] S. Goedecker, W. Hellmann and T. Lenosky, *Global minimum determination of the born-oppenheimer surface within density functional theory*, Phys. Rev. Lett. **95** 055501 (2005), doi:10.1103/PhysRevLett.95.055501.

[56] D. E. Knuth, *The Art of Computer Programming, Volume 3: (2nd ed.) Sorting and Searching*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA (1998).

[57] Message Passing Interface Forum, *MPI standards*, URL `http://www.mpi-forum.org`.

Danksagung

An dieser Stelle möchte ich mich besonders bei Joost VandeVondele bedanken, der mich während meiner Diplomarbeit in Zürich betreut und unterstützt hat. Außerdem möchte ich mich bei Frank Noé und Knut Reinert dafür bedanken, dass sie diese interdisziplinäre Arbeit ermöglicht haben. Weiterhin möchte ich mich bei Stefan Goedecker, Florian Schiffmann, Daniel Sebastiani und Christoph Wehmeyer für die anregenden Diskussionen bedanken. Bei Sebastian Ohl und Samuel Andermatt möchte ich mich für ihr Korrekturlesen bedanken. Ganz besonders möchte ich mich bei Katharina Mellert und Kai Hünnefeld für ihre Unterstützung bedanken. Nicht zuletzt danke ich auch meinen Eltern, denn ohne sie wäre es mir niemals möglich gewesen neben der Physik auch die faszinierende Welt der Informatik zu ergründen.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Zürich, den 1. Januar 2014