GLOBAL OPTIMIZATION OF LENNARD-JONES ATOMIC CLUSTERS

GLOBAL OPTIMIZATION OF LENNARD-JONES ATOMIC CLUSTERS

By

Ellen Fan, M.Sc.

A Thesis

Submitted to the School of Graduate Studies

in Fulfillment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (2002) McMaster University

COMPUTING & SOFTWARE Hamilton, Ontario

TITLE: Global Optimization of the Lennard-Jones Atomic Cluster

AUTHOR: Ellen Fan

SUPERVISOR: Dr. Tamás Terlaky

NUMBER OF PAGES: 123

ABSTRACT

The Lennard-Jones problem is defined as finding the coordinates of a system in three-dimensional Euclidian space that represents a potential energy minimum. Lennard-Jones potential energy plays a key role in determining the stability of crowed and highly branched molecules such as proteins. The main difficulty in solving this problem arises from the fact that the objective function is a non-convex and highly nonlinear function of many variables with a large number of local minima. Due to its importance, this problem has attracted many researchers from diverse fields.

In this thesis, we propose a deterministic global optimization approach, which is a combination of direct search methods with local heuristics, in the aim of finding the global optimal energy configuration of Lennard-Jones micro-clusters. With our methods, global optima are located for micro-clusters of 2 to 30 atoms. The results in this thesis are compared to the

best known results from the literature and to the ones based on LGO branch and bound and random sampling methods, as well as to the ones based on gradient-based methods. The proposed approaches were implemented in MATLAB. We provide a full implementation of the Nelder-Mead simplex method and the DFO algorithm.

ACKNOWLEDGMENTS

This work has been done in the Advanced Optimization Laboratory in the Department of Computing and Software at McMaster University. I gratefully acknowledge Dr. T. Terlaky, the leader of the laboratory, my supervisor, who spent many hours of his time on helping and teaching me. I thank him for the opportunity he gave to me doing this study and for all the lessons he gave to me.

I would like to thank the members of my committee, Dr. N. Nedialkov, Dr. J. Peng and Dr. T. Terlaky, for their kind and careful examination of this thesis and for many helpful suggestions.

I am also indebted to Dr. J. Pintér, who provided his software LGO and taught me to use it.

I owe special thanks to Dr. R. Janicki, Graduate Advisor of the Department CAS at the time I applied for a graduate study, for ever agreeing to accept

me into the graduate program.

Finally, I thank all members of the optimization group for their kind help and for the enjoyable and friendly time they gave to me.

Contents

List of Figures

List of Tables

List of Symbols

Symbols used throughout this thesis

$(\cdot)^T$	Transpose
H	A matrix to approximate Hessian
α	Step length in line search
$\arg\min$	Argument of a minimization problem at optimum
λ	Lagrange multiplier
$\min$	Minimize
∇	First order derivative
∇^2	Second order derivative
$\tilde{e}$	Approximate convergence sequence
$\|\cdot\|$	2-norm
e	Convergence sequence
f	Objective function

Symbols used throughout this thesis (cont.)

$l(\cdot)$	Linear approximate function
n	Dimension of the problem
$q(\cdot)$	Quadratic approximate function
s	Search direction
x	A vector

Superscripts used throughout this thesis

$*$	Optimal
k	Iteration counter

Symbols related to LJ problem

N	Number of the atoms
V	Potential energy of a LJ cluster
Y	Coordinates in R^{3N}
v	Potential energy of a pair of atoms
r	Distance between 2 atoms
y	Coordinates in 3D

Superscripts in Nelder-Mead simplex method

e	Expansion
ic	Inside contraction
new	New
oc	Outside contraction
r	Reflection

Symbols related to DFO

$N_i^l(\cdot)$	A basis of NFP
$Q(\cdot)$	Quadratic model
X	Interpolation point set
Δ	Trust region radius
ξ	Quadratic model coefficients
$\Phi(\cdot)$	Coefficient matrix
$\check{x}$	Optimal solution for the trust region maximization subproblem
$\hat{x}$	Optimal solution for the trust region minimization subproblem
$\phi(\cdot)$	A monomial of a polynomial
ε_Δ	Minimum value of the trust region radius
ε_{fun}	Function reduction tolerance

Superscripts in DFO

b Best

c Center of a trust region

d Drop

f Furthest

p Points index in NFP

Set notations

$\cap$ Intersection

$\cup$ Union

$|\cdot|$ Length

$\setminus$ Minus

List of Abbreviations

3D	Three Dimension
BFGS	Broyden-Fletcher-Goldfarb-Shanno
B&B	Branch and Bound
DFO	Derivative Free Optimization algorithm
DFP	Davidon-Fletcher-Powell
LGO	Lipschitz Global Optimization software package
LJ	Lennard-Jones
MATLAB	Matrix Laboratory, a language for technical computing
NFP	Newton Fundamental Polynomial
PDS	Parallel Direct Search
PES	Potential Energy Surface
SA	Simulated Annealing
SR1	Symmetric Rank One
VRML	Virtual Reality Modelling Language

Chapter 1

Introduction

One of the simplest to describe, yet most difficult to solve, problems in computational chemistry is the determination of molecular conformation. A molecular conformation problem can be described as finding the global minimum of a suitable potential energy function, which depends on relative atom positions. Progress toward solution techniques will facilitate drug design, synthesis and utilization of pharmaceutical and material products.

The success of computational methods to solve such kind of problems hinges on two factors: (1) a suitable potential energy function to predict the native states of the system as the global minimizer of the potential energy function and (2) the available minimization algorithms that can be used to

locate efficiently the global minimizer of the potential energy function.

Molecules consist of electrons and nuclei. Most applications of quantum chemistry separate the motion of the nuclei from the motion of electrons. This treatment is called Born-Oppenheimer approximation. This approximation results in a model of nuclei moving on a potential energy surface, with electrons adjusting instantly to changes in nuclear positions. Nuclear motion is constrained by the interaction of nuclei and electrons. At any fixed positions of the nuclei, the potential energy is the sum of repulsion among the positively charged nuclei and attractions arising from the electrons. Electronic energy can be computed by solving the quantum mechanical Schrödinger equation [HP63]. The most important structures are stable, equilibrium molecular geometries and transition states. The equilibrium geometry of a molecule (bond lengths and angles) describes the coordinates of a deep minimum on a potential energy surface.

The methods of quantum chemistry are quite suited to predict the geometric, electronic and energy features of known and unknown molecules. However, "it remains too expensive in terms of computer time and nearly intractable, even at the simplest, semi-empirical level, for many organic molecules or biological macromolecular structures" [DW96] . Therefore, in-

creased interest has focused on models that are able to give quickly an energy favorable conformation for large systems. *Molecular mechanics* or *empirical force field* methods are techniques that play an important role in the research of molecular conformation.

1.1 Molecular Mechanics

Basically, molecular mechanics treats molecules as being composed of masses and springs, where masses represents the atoms and springs represents bonds. In Figure 1.1, we show the visualization of different deformations (principally the torsion, the bond stretching and the angle bending) operating on bonds of a simple molecule.

The deformation due to interaction between two non-bonded atoms represents the action of Van der Waals attraction, steric repulsion and electrostatic attraction-repulsion on these two atoms.

The goal of molecular modelling is to predict the energy associated to a given conformation of a molecule. The energy of a target molecule depends on the relative positions of its atoms. This energy can be approximately

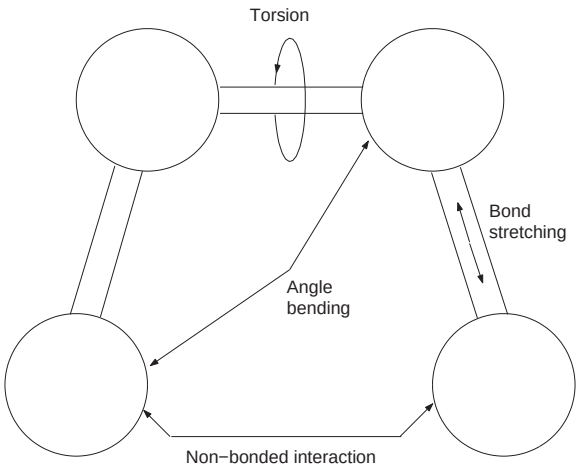

Figure 1.1: Different deformation of a molecule: torsion, bond stretching, angle bending and non-bonded interaction.

estimated by the sum of several contributions [FZ99]:

$$V_{MM} = V_b + V_\theta + V_\tau + V_{nb} + (\textit{specific terms}), \tag{1.1}$$

where V_{MM} is often referred to as the steric energy or potential energy. It corresponds to the energy difference between the real molecule and a hypothetical molecule in which all structural values, such as bond lengths and bond angles are exactly ideal (equilibrium values). In equation (1.1),

V_b represents the bond energy, describing the compression or the extension of a bond from its equilibrium length;

V_θ represents the angle bending energy, and is the function of bond curve in

respect to its equilibrium value;

V_τ is the torsion energy;

V_{nb} is the interaction energy between two non-bonded atoms; and

specific terms could be out of plane bending, electrostatic interactions and possible hydrogen bonding.

Bond stretching The bond stretching contribution is represented by Hooke's law. It measures the energy due to the variation of bond length after extension or compression from their equilibrium values:

$$V_b = \frac{1}{2}\sum_{i=1}^{n} k_{r,i}(r_i - r_i^0)^2, \tag{1.2}$$

where

$k_{r,i}$ is the bond force constant;

r_i is the bond length;

r_i^0 is the bond length at equilibrium position; and

n is the total number of bonds in the molecule.

The parameters $k_{r,i}$ and r^0 are invariant, depending only on the type of each pair of connected atoms. Equation (1.2) is a rough approximation of bond energy. Alternatively, a Morse potential can be used to describe more

precisely the bond stretching energy due to the variation of a bond length:

$$V_b = \sum_{i=1}^{n} D(1 - \exp(-a(r_i - r_i^0)))^2,$$

where D and a are parameters characterizing the bond. The use of such a potential seems to be useful for elongated hydrogen bonds, which otherwise tend to dissociate [All85].

Angle bending The angle bending potential determines the energy quantity resulted by the angle variation between two adjacent bonds based on an equilibrium bond angle. In the case of harmonic approximation, this is equally derived from Hook's law:

$$V_\theta = \frac{1}{2} \sum_{ij} k_{\theta,ij}(\theta_{ij} - \theta_{ij}^0)^2, \tag{1.3}$$

where

$k_{\theta,ij}$ is the force constant;

θ_{ij} is the bond angle between 3 atoms; and

θ_{ij}^0 is the bond angle at equilibrium position between 3 atoms.

Torsion The torsion energy represents the energy modification of the rotation of the molecule around a bond. The most common expression which permits to describe the evaluation of the molecule energy as the function of

torsion angle is the Fourier series:

$$V_\tau = \frac{1}{2}\sum_i A_{i,n}[1 + \cos(n\tau_i - \Phi)], \tag{1.4}$$

where

$A_{i,n}$ is the force constant which controls the curve amplitude;

Φ is the phase;

n is the periodicity of $A_{i,n}$; and

τ_i is the torsion angle.

Torsion energy is in fact a correction of different energy terms rather than a physical process. It represents the energy quantity that should be added to or subtracted from the summation of $V_b + V_\theta + V_{nb}$ in order to obtain the geometry in good agreement with an experiment or with the geometry that is deduced from quantum chemical calculations.

Energy of non-bonding interactions Interaction between two non-bonding atoms is the principal cause of steric hindrance, which play an important role in the molecular geometry. The energy of non-bonding interactions is the sum of energies of all non-bonding atoms acting between them. It includes the energy of Van der Waals interaction, electrostatic energy and induction energy three terms.

The term Van der Waals interaction is generally described by the LJ

potential:

$$V_{vdw} = \sum_{i<j} \left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} \right), \tag{1.5}$$

where

r_{ij} is distance between two non-bonding atoms i and j; and

A_{ij} and B_{ij} are Van der Waals constants.

The summation is taken over all non-bonded pairs of atoms (i, j).

These expressions involve two terms:

1. An attractive part, corresponding to induced dipole-induced dipole interaction, proportional to r_{ij}^6, where r_{ij} is the distance between the two atoms i and j.

2. A repulsive part, corresponding to London dispersion [HP63] terms and rapidly growing as the distance is getting shorter.

For a given geometrical arrangement of the atoms in a molecule system, the steric energy, due to distortions of bond lengths and angles with respect to the reference values and Van der Waals interaction, can be calculated according to the potential energy function. To determine the actual equilibrium geometry, this steric energy with respect to all internal degrees of freedom

must be minimized.

The term of electrostatic energy increases with the polarity of chemical bonds. It can be expressed using Coulomb potential [HP63]. The induction energy is the consequence of the distortion of electronic distribution, which depends on the electric field created by other molecules, and generates induced electric moments.

Bond lengths and bond angles are usually available from existing structural information, i.e., from X-ray crystallography. Bond stretching parameters can be directly derived from vibrational force constants. The coefficients of the torsion barriers can be estimated from barrier heights obtained through microwave spectroscopy, thermodynamic studies, or far infrared and Raman spectroscopy. More difficult is the evaluation of the Van der Waals interaction, a crucial point since these interactions are important in determining the stability of crowded or highly branched molecules such as peptides.

1.2 The LJ Cluster Problem

As described in Section 1.1, the Van der Waals potential characterizes the contribution of the non-bonded pairwise interactions between atoms. It is

generally described by the LJ potential function, as it was introduced by Equation (1.5). The LJ potential is a key part of many empirical energy models, including all commonly used energy functions for proteins. A system containing more than one atom, whose Van der Waals interaction can be described by LJ potential is called a LJ cluster.

In our work, we employ the scaled LJ pair potential:

$$V = \sum_{i<j} \left(\frac{1}{r_{ij}^{12}} - \frac{2}{r_{ij}^{6}} \right) \tag{1.6}$$

The LJ potential function for a single pair of neutral atoms is a simple unimodal function. This is illustrated by Figure 1.2.

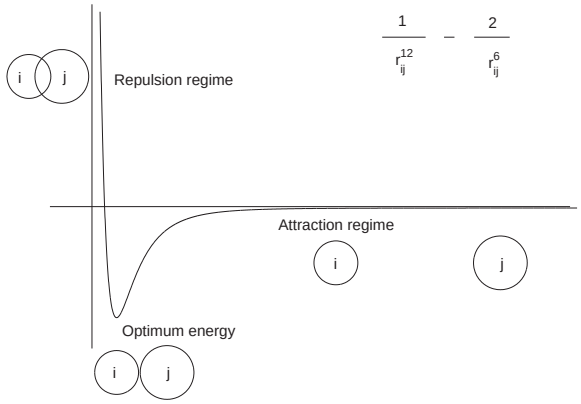

Figure 1.2: LJ potential of a single pair of atoms.

It is easy to find the overall minimum of this function that is assumed at 1 with energy -1. In a complex system, many atoms interact and we need

to sum up the LJ potential functions for each pair of atoms in a cluster. The result is a complex energy landscape with many local minima.

Given a cluster, e.g., a three-atom cluster, each atom interacts with two other atoms. The LJ potential can be written as :

$$\begin{aligned} V &= \left(\frac{1}{r_{12}^{12}} - \frac{2}{r_{12}^{6}}\right) + \left(\frac{1}{r_{13}^{12}} - \frac{2}{r_{13}^{6}}\right) + \left(\frac{1}{r_{23}^{12}} - \frac{2}{r_{23}^{6}}\right) \\ &= \sum_{1 \le i < j \le 3} \left(\frac{1}{r_{ij}^{12}} - \frac{2}{r_{ij}^{6}}\right) \end{aligned}$$

For illustration, we fix one atom's position and let the others move around the fixed one. The plot of the Potential Energy Surface (PES) is illustrated in Figure 1.2 (the peaks were cut at level 1). A cluster of more than 20 atoms has many of local minima along its LJ PES.

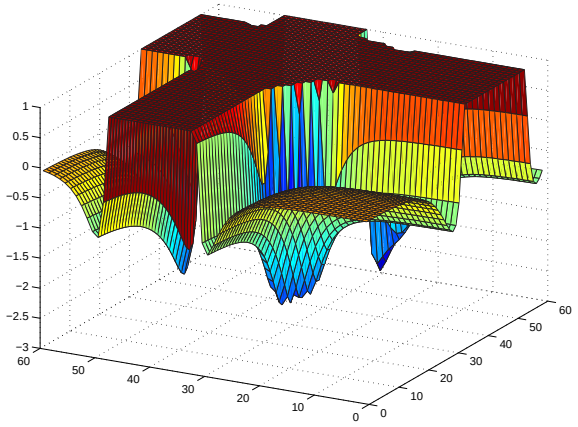

Figure 1.3: A two dimensional section of the LJ PES of a cluster of 8 atoms.

Note that, we use in the summation $i < j$, because we need to account

the interaction between two atoms only once. If one uses $i \neq j$, the total energy must be divided by 2. In other words, the LJ potential function is *partially separable*[1]. The partially separability of the LJ potential implies that, if a single atom or molecule in a cluster is moved, the potential energy can be re-evaluated cheaply at a cost that is only $\frac{2}{N}$-th of the cost of a total function evaluation, where N is the total number of atoms or molecules in the cluster. This is due to the fact that the potential function composed as the sum of pairwise interactions between atoms or molecules.

Given a cluster of N atoms, the LJ cluster problem is to *find the relative position of atoms in the three-dimensional Euclidean space that represent a potential energy minimum.*

1.3 Formulation

Let $y^i = (y_1^i, y_2^i, y_3^i)^T$ represent the coordinates of atom i in the three-dimensional Euclidean space. Let $Y = ((y^1)^T, \ldots, (y^N)^T)^T$, where N is the number of atoms in the cluster.

[1]A function that is the sum of functions, each of which only involves a disjoint subset of the variables, is called partially separable.

The LJ potential of a pair of atoms (i, j) is

$$v(r_{ij}) = \frac{1}{r_{ij}^{12}} - \frac{2}{r_{ij}^{6}},$$

where $r_{ij} = \|y^i - y^j\|$.

The LJ cluster problem described in the previous section can be formulated in the coordinate space as follows:

$$\begin{aligned} \min_{Y \in R^{3N}} V(Y) &= \sum_{i<j} v\left(\|y^i - y^j\|\right) \\ &= \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left(\frac{1}{||y^i - y^j||^{12}} - \frac{2}{||y^i - y^j||^{6}} \right), \end{aligned}$$

where y^i and y^j represent the coordinates of the i-th and the j-th atom, respectively.

As it was illustrated by Figure 1.2, for a single pair of neutral atoms, the overall potential energy minimum is reached when the distance between two atoms is 1. When this distance approaches zero, the potential tends to infinity. When an atom is far away from the system, its contribution to the total potential becomes almost zero. Due to these observations, it is reasonable to expect that at the optimal solution of the LJ cluster problem all atoms in R^3 are close to unit distance to each other.

It has been shown that the complexity of determining the global minimum energy of the LJ cluster belongs to the class of NP-hard problem [WV85].

In other words, there is no known algorithm that can solve this problem in polynomial time [GJ79]. An algorithm that is suitable for solving this kind of problems would appeal to geometric intuition, and probably the procedure would require widely spaced measurements, in order to smooth out the variations of the valleys and hills. This motivated us to investigate direct search methods in the global optimization of LJ potential energy functions.

Various computational approaches for the LJ cluster problem can be found in the literature, ranging from mathematical programming models [GKS97], to genetic algorithms [DTMH96b] and to Monte Carlo sampling method [WD97]. Good results have been obtained by coupling some of these methods. An example is the so-called Langevin dynamics method, proposed by Biswas and Hamann [BH86]. This minimization method is a combination of simulated annealing and gradient techniques, and has been proved to be efficient to minimize the total energy of small systems.

The main difficulty in solving the LJ minimization problem arises from the fact that the objective is a non-convex function of many variables with a large number of local minima. This non-convexity makes it very difficult to find global optimal solutions. In this thesis, we propose a deterministic global optimization approach that is a combination of direct search methods with

local heuristics, in the aim of finding the global optimum energy configuration of LJ micro-clusters.

An application of direct search methods to molecular geometry optimization has been reported by Meza and Martinez [MM94] earlier. They have compared the PDS method of Dennis and Torczon [DT91], genetic algorithms and simulated annealing using LJ potentials. They conclude that PDS could also be used in conformational searching and show that it performed as well as genetic algorithms and substantially better than simulated annealing for large molecules.

An outline of this thesis is as follows. First we will describe some of the non-linear optimization methods including global optimization approaches, derivative-based methods and direct search methods in Chapter 2. Then, we give our algorithms for the LJ cluster problem in Chapter 3. Implementation issues are discussed in Chapter 4. Some experimental results are reported in Chapter 5. Chapter 6 contains conclusions and suggestions for further research.

Chapter 2

Nonlinear Optimization Methods

Methods used for finding the global minimum of LJ potential are mostly based on search heuristics, that include mimicking of physical phenomena, random searches, lattice optimization combines with local optimization approaches. In this chapter, we review some general optimization techniques including global optimization approaches, derivative-based methods and direct search methods. The algorithms that are used in our approach are described in detail.

2.1 Global Optimization Approaches

Global optimization approaches can be classified as probabilistic or deterministic. Methods such as simulated annealing and genetic algorithms are probabilistic methods. Gradient, Quasi-Newton [Fle87] and Branch and Bound methods [Pin96] are deterministic.

2.1.1 Simulated Annealing

As its name implies, the Simulated Annealing (SA) exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a general system. If a physical system is melted and then cooled slowly, the entire system can be made to produce the most stable (crystalline) arrangement, and not get trapped in a local minimum [WC90].

The SA algorithm was first proposed by Metropolis et al. [MRR$^+$58] as means to find the equilibrium configuration of a collection of atoms at a given temperature. The connection between this algorithm and mathematical minimization was first noted by Pincus [Pin70], but it was Kirkpatrick et al. [KGV83] who used it as the basis of an optimization technique for combinatorial (and other) problems.

SA's major advantage over other methods is its ability to avoid being trapped at a local minima. The algorithm employs a random search, which not only accepts changes that decrease the objective function f, but also some changes that increase it. The latter are accepted with a probability

$$p = \exp\left(-\frac{\delta f}{T}\right)$$

where δf is the increase in f and T is a control parameter, which by analogy with the original application is known as the system "temperature" irrespective of the objective function involved.

Let us describe briefly how SA works. Given a function to optimize, and some initial values for the variables, simulated annealing starts at a high, artificial, temperature. While cooling the temperature slowly, it repeatedly chooses a subset of the variables and changes them randomly in a certain neighborhood of the current point. If the objective function has a lower function value at the new iterate, the new values are chosen to be the initial values for the next iteration. If the function value is higher, the new values are chosen to be the initial values for the next iteration with a certain probability, depending on the change in the value of the objective function and the temperature. The higher the temperature and the lower the change, the more probable the new values are chosen to be the initial variables for the

next iteration. Throughout this process, the temperature is decreased gradually, until eventually the values do not change anymore. Then, the function is presumably at its global minimum. Since we can always choose a higher temperature to start, the temperature is never increased.

S.R. Wilson et al [WCMS88] applied this method to locate the global minimum energy conformation of flexible molecules. At a given temperature, a number of conformers in equilibrium are simulated. At each iterate, a rotatable bond is selected randomly to be rotated a random number of degrees and the resulting change in energy of the molecule, ΔE, is calculated according to a selected force field. If $\Delta E \leq 0$, the rotation is accepted, and the new conformation is used as the starting point for the next step. If $\Delta E > 0$, the acceptance is treated probabilistically. The probability is $p(\Delta E) = \exp(-\Delta E/kT)$, where k is a constant. The lowest energy conformation is kept and is updated whenever a lower one is found. The global energy minimum conformation is obtained by choosing appropriate " cooling schedule".

2.1.2 Genetic Algorithms

Genetic algorithms are optimization techniques derived from the principles of evolutionary theory. They contain a population of individuals, each of them has a known fitness. The population is evolved through successive generations until a stoping criteria is satisfied. A genetic algorithm represents points in the search space by a vector of discrete (typically) bit values. A new child is produced by combining parts of the bit vector from its parent. This is analogous to the way that chromosomes of DNA (which contains the inherited genetic material) are passed to children in nature.

One of the main interests in genetic algorithms is their application to difficult, multi-extreme optimization problems. For instance, Deaven et al. [DTMH96a] applied this technique to the problem of determining the lowest energy configurations of a collection of atoms, including LJ clusters. They start with a population of candidate structures, for example a population of p clusters. These clusters are "mated" to form new child clusters by choosing a random plane passing through the center of mass of each "parent" cluster, then cutting the parent clusters in this plane, and assembling the "child" from the two halves. Each child produced in this way then apply a local search procedure. The process is repeated until several generations

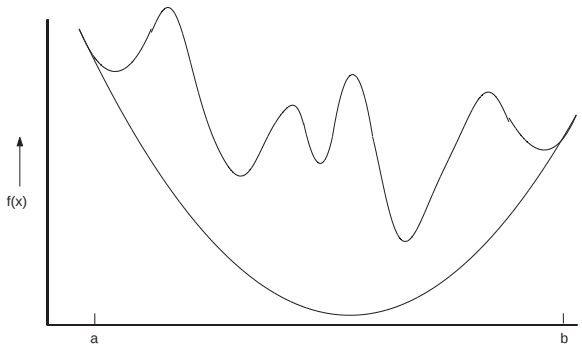

Figure 2.1: Continuation approach using transformation

go by with no further reduction in energy, indicating that the best known global minimum structure has been located. Deaven's algorithm was able to locate most of the best known global minimum energy structures of the clusters $N = 2, \ldots, 100$ reported in the literature. In addition, some lower energy configurations, clusters of $N = 38, 65, 69, 76, 88$ and 98, have been found with their method.

2.1.3 Continuation Approach

In the continuation approach for global optimization, the original function is gradually transformed into a smoother function with fewer local minimizers. This is illustrated in Figure 2.1.3, for a one-dimensional non-convex objective function.

Consider for example the problem

$$\min f(x),$$

where $f(x) : R^n \rightarrow R$ is a non-convex[1] objective function with many local minima. We transform this problem into

$$\min \quad h_\mu(x) = f(x) + \mu q(x),$$

where $q(x)$ is strictly convex, and $\mu \geq 0$. Thus $h_\mu(x)$ is convex for a sufficiently large μ. Let $x(\mu) = \arg\min_x \{h_\mu(x)\}$ be the minimizer of $h_\mu(x)$ for a fixed μ. Then $x(\mu)$ approaches $x^* = \arg\min \{f(x)\}$ as $\mu \rightarrow 0$. Here, the objective is convexified by adding a strictly convex term $\mu q(x)$, and then gradually changed back to the original when $\mu \rightarrow 0$. This is not simulated annealing, but μ might be considered as a similar control parameter as the temperature T in the simulated annealing.

The main approaches of transforming a function into a smoother function include the diffusion equation method of Piela, Kostrowicki, and Scheraga [PKS89], the packet annealing method of Shalloway [Sha92], and the

[1]A function $f : C \rightarrow R$ defined on a convex set C is called convex if for all $x^1, x^2 \in C$ and $0 \leq \tau \leq 1$ s.t.

$$f(\tau x^1 + (1-\tau)x^2) \leq \tau f(x^1) + (1-\tau) f(x^2).$$

If strict inequality holds for all $x^1 \neq x^2$ and $0 < \tau < 1$, then the function f is called strictly convex.

effective energy simulated annealing method of Coleman, Shalloway, and Wu [CSW94]. Other transformations used in molecular conformation problems are reviewed by Straub [Str94].

Although Moré and Wu [MW95] claimed that their algorithms based on the continuation approach for global optimization can be used to solve a distance geometry problem with nearly 100 percent probability of success, the continuation process cannot succeed on an arbitrary function. In particular, the transformation may eliminate tall, narrow valleys (hills) while the global minimizer (maximizer) may lie in one of these valleys (hills).

In a similar way, Locatelli and Schoen [LS02] considered the continuous approach for the global optimization of the LJ clusters. They first minimize a modified convex potential function, which, is related to the LJ potential

$$h(r_{ij}) = \frac{1}{r_{ij}^{2p}} - \frac{2}{r_{ij}^{p}} + \mu r_{ij} + \beta(\max\{0, r_{ij}^2 - D^2\})^2,$$

where p, μ, β and $D \geq 0$. The value of p can be $3, 4$ or 5. D is an underestimate of the diameter of the cluster; μ and β are parameters of the penalty terms. These parameters must be set carefully with heuristic. The local minimum of this modified potential is then used as a starting point for a local optimization of the LJ potential function. Some difficult cluster conformation have been rediscovered with different parameter settings. However, two

pitfalls limit the practical use of their approaches. The first is the lack of a general rule to choose a set of parameters that is sufficiently good for a large range of clusters. The other is that they use a random generation mechanism in their implementation. Therefore, some 10 000 random trials [LS02] may be needed to rediscover one cluster conformation.

2.1.4 Branch and Bound Method

Branch and bound (B&B) is an approach to search for an optimal solution by searching only a part of the search space, while the derived bounds on the objective function guarantee that no optimal solution exist on the excluded or pruned parts of the search space. B&B guarantees to find a global minimizer with a desired accuracy after a predictable number of steps. The basic idea is that the set of feasible solutions is branched or partitioned into many simpler (smaller) subsets and an effort is made to search the best feasible solution or compute a lower bound of the objective function on the subsets. Each subset will be the set of feasible solutions of a subproblem. The associated subproblem is fathomed if one of the following cases occurs.

1. The best feasible solution in that subset is found;

2. It is discovered that the subset is empty;

3. Based on the bounds, it is proved that no optimal solution in that subset exists.

If the subproblem is not fathomed, that subset may again be partitioned into smaller subsets and the same process is repeated on them. The B&B approach computes and uses both lower and upper bounds for the objective value. A lower bound can be obtained by relaxing the problem. Computing a good lower bound is an essential component of B&B method. Otherwise, the B&B approach may degenerate into searching the whole space and become impractical. An upper bound of the optimal value is the current best solution value found in a subset. If the lower bound in a subproblem is worse than an upper bound already obtained in another subproblem, then the first subproblem will never contain a better solution than the current solution. Therefore, it needs not to be explored further and can be pruned.

In continuous global optimization, the subproblems are created by dividing the feasible area into different parts. The most straightforward way to do so is by dividing the range of a variable into smaller intervals. Therefore, the complexity of branching may exponentially increase with dimension. For example, if [0,1] is the range of n variables, by dividing the [0,1] range of each variable into two subintervals result in 2^n smaller parts.

J.D.Pintér [Pin95] applied this strategy for global optimization by partitioning the feasible set and by deriving bounds based on the concept of Lipschitz continuity. A function $f(x)$ is said to be Lipschitz continuous if there exists a constant L, called Lipschitz constant, such that

$$\mid f(x^2) - f(x^1) \mid \leq L \parallel x^2 - x^1 \parallel$$

for all pairs of points x^1 and x^2 in the domain of $f(x)$. An optimization problem is Lipschitz continuous, if both the objective function and all the constraint functions are Lipschitz continuous. For one dimensional differentiable functions, this means that there is an upper bound L on the absolute value of the derivatives, which can be used to derive bounds on function values.

As an example, let us consider the one-dimensional non-convex objective function, as given on Figure 2.2, which has to be minimized over the interval $[a, b]$. Suppose that at some stage during the search, the objective function is evaluated at the points p_1 through p_{10}. Then, it is known that the objective function is underestimated by the sawtooth curve as indicated. The problem can therefore be reduced to 9 smaller sub-minimization problems over the intervals $[p_1, p_2], [p_2, p_3], \ldots, [p_9, p_{10}]$, and in each interval a lower bound on the function is obtained by the minimum of the sawtooth. Since the computed

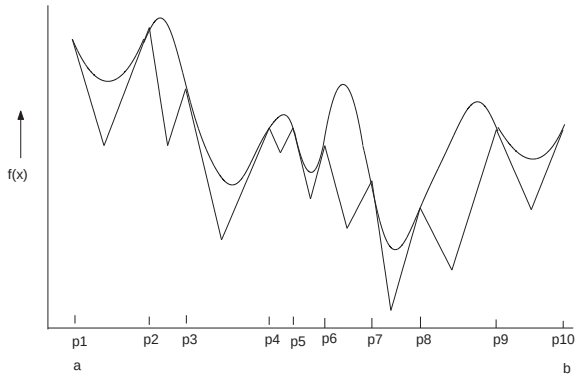

Figure 2.2: B&B using Lipschitz continuity.

objective function at point p_8 is better than the sawtooth-minimum in the intervals $[p_1, p_2], [p_2, p_3]$, $[p_4, p_5]$, $[p_5, p_6]$ and $[p_9, p_10]$, these intervals can be pruned from the search. Lipschitz optimization algorithms contain rules how to select the next point to be evaluated and how to partition the subsets, so that the partitioning is as efficient as possible, and large parts of the set of feasible solutions can be pruned as early as possible.

2.2 Derivative-Based Methods

In this section, we consider the unconstrained minimization problem

$$\min \quad f(x),$$

where $x \in R^n$, and $f : R^n \to R$. A large variety of algorithms have been developed to solve unconstrained optimization problems. Methods based only on function value comparison are called *direct search* methods. Typically, these methods are used for problems in which f is not (or not given explicitly as) a smooth function or the derivative information about f is not available. Derivative-based methods are used for problems in which f is smooth and the derivatives are easy to calculate.

For an n-dimensional unconstrained minimization problem, in the general structure of derivative-based algorithms, we distinguish two classes: one is the approach using line search (step-length-based), the other is the restricted step or the trust region approach.

2.2.1 Line Search Based Algorithms

First, we describe the general frame of line-search based algorithms. Then, we discuss its ingredients in more detail.

Let x^0 be a starting point. A line search based algorithm can be outlined as follows:

1. *Initialization.* $k = 0$.

2. *Test for stopping criteria.* If a *stopping criterion* is satisfied, stop with

the solution x^k.

3. *Determine a search direction.* Compute a non-zero vector s^k representing the search direction, a descent direction of f from x^k.

4. *Find a step length.* Compute a certain positive step length α_k, such that $f(x^k + \alpha_k s^k) < f(x^k)$.

5. *Update the iterate.* Let $x^{k+1} = x^k + \alpha_k s^k$, update $k = k + 1$, and return to Step 2.

Stopping Criteria

In nonlinear optimization, algorithms mostly can not find an exact optimal solution in a finite number of steps, but compute a sequence of approximate solutions that get closer and closer to an optimal solution. Therefore, we use a convergent sequence that defined below as criteria to stop an algorithm.

Assume that we have a sequence of points x^k converging to a solution x^*. A convergent sequence e^k that is used to measure the relative distance to optimality can be defined with respect to f

$$e_f^k = \frac{|f(x^k) - f(x^*)|}{1 + |f(x^k)|},$$

or to x

$$e_x^k = \frac{\|x^k - x^*\|}{1 + \|x^k\|}.$$

Since x^* is usually not known, in practice, we measure the progress of the algorithm using the approximate sequences $\tilde{e}^k$ of e^k defined as

$$\tilde{e}_f^k = \frac{|f(x^{k+1}) - f(x^k)|}{1 + |f(x^k)|},$$

or

$$\tilde{e}_x^k = \frac{\|x^{k+1} - x^k\|}{1 + \|x^k\|}.$$

The first sequence measures the reduction of the function value at each iteration. If this reduction is small enough, we stop the algorithm. The second measures the movement of a point at each iteration. If the iterate does not move much, we stop the algorithm. Note that, it may happen that two sequences behave very differently. If the objective is very flat, the first may become very small, while the second value is still very large. On the other hand, if the objective has a very sharp valley, the second value becomes very small while the first one is still very large. In these cases, we combine two measurements as follows:

$$\tilde{e}_{xf}^k = \frac{\|x^{k+1} - x^k\|}{1 + \|x^k\|} + \frac{|f(x^{k+1}) - f(x^k)|}{1 + |f(x^k)|}.$$

With the same assumption made on page 29, we define the convergence rate cr and rate constant C as follows [Ter00]. Let

$$\lim_{k\to\infty} \frac{\| e^{k+1} \|}{\| e^k \|^{cr}} = C < \infty,$$

where e can be any of the convergent sequences e_x^k, e_f^k, e_{xf}^k or any of the approximate convergent sequences $\tilde{e}_x^k, \tilde{e}_f^k, \tilde{e}_{xf}^k$.

If $cr = 1$ and $C = 0$, the convergence is called *superlinear*.

If $cr = 1$ and $0 < C < 1$, the convergence is called *linear*.

If $cr = 1$ and $C = 1$, the convergence is called *sublinear*.

If $cr = 1$ and $C > 1$, then the sequence diverges.

If $cr = 2$, the convergence is called *quadratic*.

Search Direction

Derivative-based methods use first-order and/or second-order partial derivatives of the function to generate the search directions at the successive iterations. Therefore, we need to assume that f is a twice continuously differentiable function. They are usually based on linear or quadratic Taylor series approximations of the function f.

1. Steepest Descent Direction

The first-order Taylor expansion around x^k of f gives:

$$f(x^k + s^k) \approx l(x^k) = f(x^k) + (s^k)^T \nabla f(x^k).$$

The *steepest descent method* searches a direction s^k such that $l(x^k)$ is minimized. Assuming that the length of s^k is normalized to $\|s^k\| = \|\nabla f(x^k)\|$, this leads to minimizing the term $\nabla f(x^k)^T s^k$ over a sphere, which leads to the search direction

$$s^k = -\nabla f(x^k),$$

called a *steepest descent direction*.

2. Newton Direction

The second-order Taylor expansion of f gives:

$$f(x^k + s^k) \approx q(s^k) = f(x^k) + (s^k)^T \nabla f(x^k) + \frac{1}{2}(s^k)^T \nabla^2 f(x^k) s^k.$$

A search direction s^k is selected so that it minimizes $q(s^k)$. By setting the gradient of $q(s^k)$ equal to zero, one obtains that the search direction can be computed by solving the system

$$\nabla^2 f(x^k) s^k = -\nabla f(x^k) \tag{2.1}$$

The direction

$$s^k = -(\nabla^2 f(x^k))^{-1} \nabla f(x^k)$$

is called the *Newton direction*. As we can see, to have a unique solution of equation (2.1), the Hessian $\nabla^2 f(x^k)$ must be non-singular. Furthermore, to have a descent direction, $\nabla^2 f(x^k)$ has to be positive definite.

3. Quasi-Newton Direction

Quasi-Newton methods are based on the same idea of building up curvature information, as the iterations of the decent method proceeds. In Newton methods, the user must supply the Hessian while in Quasi-Newton methods, only gradient information is used. The Hessian is approximated by a symmetric positive definite matrix H_k, which is corrected or updated from iteration to iteration. The search direction s^k is given as

$$s^k = -H_k \nabla f(x^k).$$

Line Search

In line search based methods, each iteration computes a search direction s^k and then decides how far to move along this direction. The iteration is given

by

$$x^{k+1} = x^k + \alpha_k s^k,$$

where the positive scalar α_k is called the *step length*. The ideal choice of α_k would be the global minimizer of the univariate function $\phi(\cdot)$ defined by

$$\phi(\alpha) = f(x^k + \alpha s^k),$$

where $\alpha > 0$.

There are a number of methods that find the global minimizer α^*, when the function is *unimodal*, which means that there is $\alpha^* \in [\alpha^{min}, \alpha^{max}]$ such that the function is non-increasing on the interval $[\alpha^{min}, \alpha^*]$ and non-decreasing on the interval $[\alpha^*, \alpha^{max}]$. Among them are golden section search, quadratic interpolation and cubic interpolation. However, in general, it is too expensive to identify this value since it requires too many function evaluations, and the function is usually not unimodal. More practical strategies perform an *inexact* line search to identify a step length that achieves adequate reductions in f at minimal cost. An often applied practical rule is the Goldstein-Armijo rule [BSS93].

The Goldstein-Armijo rule defines a range of acceptable step lengths by specifying a sufficient decrease in the objective function and curvature condition criteria for the minimal and the maximal allowed step length.

Given a starting point x^k, the first order approximation of $\phi(\alpha)$ is given by

$$f(x^k) + \alpha \nabla f(x^k)^T s^k.$$

The Goldstein-Armijo rule relates the actual function value $\phi(\alpha)$ to this approximation and considers a step α acceptable if

$$f(x^k) + \mu_1 \alpha \nabla f(x^k)^T s^k \leq f(x^k + \alpha s^k) \leq f(x^k) + \mu_2 \alpha \nabla f(x^k)^T s^k$$

for some μ_1 and μ_2 satisfying $1 > \mu_1 > \mu_2 > 0$. Typical values of μ_1 are 0.9 when the search direction s^k is chosen by a Newton method and 0.1 when s^k is obtained from a nonlinear conjugate gradient method. μ_2 is chosen to be quite small, say $\mu_2 = 10^{-4}$ [NW99].

Derivative Based Methods

Now we give some detailed description of line search based methods.

1. Steepest-Descend Method

The method of steepest descent is an old and well-known method for unconstrained minimization of a function f with continuous first-order partial derivatives. The steps of the steepest descend method can be described as follows:

Specify a starting point x_0.

For $k = 0, 1, \cdots$

If a stopping criterion (see page 29) is satisfied, stop.

Let $s^k = -\nabla f(x^k)$.

Line search $\alpha_k = \arg\min_{\alpha} f(x^k + \alpha s^k)$.

Compute $x^{k+1} = x^k + \alpha_k s^k$.

The rate of convergence of the steepest-descend method is linear. Practical experience has shown that it is not a very efficient method. The performance of the steepest descent method depends also on the efficiency and accuracy of the line search. In practice, we use inexact line search, typically, the Goldstein-Armijo [BSS93]. The total cost per iteration relies on two terms: $O(n)$ arithmetic operation to compute the gradient and the cost to do line search. If we use the Goldstein-Armijo, then it requires at most 20 function evaluations per iteration.

2. Newton Method

The basic Newton method can be described as follows:

Specify a starting point x_0.

For $k = 0, 1, \cdots$

If a stopping criterion (see page 29) is satisfied, stop.

Solve $\nabla^2 f(x^k)s^k = -\nabla f(x^k)$ for s_k.

Compute $x^{k+1} = x^k + s^k$.

The Newton method has a quadratic rate of convergence. However, to achieve such a rapid rate of convergence, it requires that the initial point is sufficiently close to a minimizer. Otherwise, if the initial point is not close to a minimizer, then the classical Newton method without line search may diverge or not converge to a minimizer. The cost associated with a single step of the Newton method includes two parts:

1. It requires $O(n)$ and $O(n^2)$ arithmetic operations to compute the gradient and the Hessian, respectively; moreover, $O(n^3)$ arithmetic operations are needed to solve the Newton system for the search direction.

2. It requires $O(n)$ and $O(n^2)$ space to store the gradient and the Hessian, respectively.

The Newton method is used in its classical form primarily for convex functions to ensure a descent search direction. On the other hand, if all

the pre-required conditions are satisfied, then it is an "ideal" method in the sense that it has a quadratic convergent rate for solving minimization problems. Many variants and approximations of the Newton method have been proposed and used to solve real world problems.

3. Quasi-Newton Method

Quasi-Newton methods may be the most widely used methods for nonlinear optimization. The general Quasi-Newton method can be described as follows.

Specify a starting point x^0, and an initial guess H^0 that approximates the Hessian.

For $k = 0, 1, \cdots$

If a stopping criterion (see page 29) is satisfied, stop.

Set $s^k = -H^k \nabla f(x^k)$

Line search along s^k giving

$x^{k+1} = x^k + \alpha_k s^k$.

Compute

$\delta^k = x^{k+1} - x^k = \alpha_k s^k$

$\gamma^k = \nabla f(x^{k+1}) - \nabla f(x^k)$.

Compute $H^{k+1} = H^k + \text{update formula}(\delta^k, \gamma^k)$.

The step length α_k is computed by an inexact line search that satisfies the Wolf or strong Wolf condition [NW99]. Some updating formulae are given as follows:

Symmetric Rank One (SR1) update formula:

$$\frac{(\gamma^k - H^k\delta^k)(\gamma^k - H^k\delta^k)^T}{(\gamma^k - H^k\delta^k)^T\delta^k}.$$

Davidon-Fletcher-Powell (DFP) update formula:

$$-\frac{H^k\delta^k(\delta^k)^T H^k}{(\delta^k)^T H^k\delta^k} + \frac{\gamma^k(\gamma^k)^T}{(\gamma^k)^T\delta^k} + [(\delta^k)^T H^k\delta^k]\omega^k(\omega^k)^T,$$

where

$$\omega^k = \frac{\gamma^k}{(\gamma^k)^T\delta^k} - \frac{H^k\delta^k}{(\delta^k)^T H^k\delta^k}.$$

Broyden-Fletcher-Goldfarb-Shanno (BFGS) update formula:

$$-\frac{H^k\delta^k(\delta^k)^T H^k}{(\delta^k)^T H^k\delta^k} + \frac{\gamma^k(\gamma^k)^T}{(\gamma^k)^T\delta^k}.$$

The advantages of the Quasi-Newton methods are:

1. Only first order derivatives are required;

2. The positive definiteness of H^k implies the descent property;

3. Only $O(n^2)$ arithmetic operations are required per iteration. Since we don't solve the linear system explicitly, but use the information from the previous iterate to calculate the search direction, the search direction can be calculated using only $O(n^2)$ arithmetic operations. The update of the Cholesky factors of H^k can be done also in $O(n^2)$ arithmetic operations.

The rate of convergence of the Quasi-Newton methods is superlinear, which is fast enough for most practical purposes. We know that the classical Newton method converges more rapidly, but its cost per iteration is higher because it requires the solution of a linear system, and it requires calculation of second derivatives.

2.2.2 Trust Region Methods

Trust-region methods, like Newton's method, make explicit reference to a "model" of the objective function. For Newton's method, this model is a quadratic model derived from the Taylor series of f about the point x^k.

$$f(x^k + s) \approx q_k(s) = f(x^k) + \nabla f(x^k)^T s + \frac{1}{2} s^T \nabla^2 f(x^k) s. \qquad (2.2)$$

In line search based methods, when a minimizing direction s is found, the method proceeded with a line search. In trust-region methods, the model function is minimized without line search, but under the restriction that

$$\|s\| \leq \triangle_k \tag{2.3}$$

for some positive $\triangle_k$, which is the trust region radius. This serves to limit the size of the step taken from x^k to x^{k+1},

$$x^{k+1} = x^k + s^k,$$

where s^k is k-th iteration search direction. The value of $\triangle_k$ is adjusted repeatedly based on the agreement or discrepancy between the model function $q_k(s^k)$ value and the objective function $f(x^k + s^k)$ value. If the agreement is good, then the model can be trusted and $\triangle_k$ increased. If not, then $\triangle_k$ is decreased, or $\triangle_k$ remains unchanged if neither agreement nor discrepancy is strong. The measurement of the agreement or discrepancy is discussed later.

Thus, at each step, we need to find the minimizer of the approximation $q_k(s)$. The strategies that are used to find such minimizers are surveyed in [NW99]. We describe here a strategy of Moré and Sorensen that finds a "nearly exact" solution of the approximate $q_k(s)$ [NW99].

For the 2-norm, the minimization problem of (2.2) with constraint (2.3)

is equivalent to the unconstrained minimization of

$$\nabla f(x^k)^T s + \frac{1}{2} s^T (\nabla^2 f(x^k) + \lambda I) s,$$

where $\lambda \geq 0$ is a scalar, called the Lagrange multiplier for the constraint $\|s\| \leq \triangle_k$ and $\nabla^2 f(x^k) + \lambda I$ needs to be positive semi-definite, finally

$$\lambda(\triangle_k - \|s\|) = 0.$$

At iteration k of a trust region method, s^k is the solution of the equation

$$(\nabla^2 f(x^k) + \lambda I) s^k = -\nabla f(x^k)$$

with either $\lambda = 0$ and $\|s^k\| < \triangle_k$ or $\lambda > 0$ and $\|s^k\| = \triangle_k$. If $\nabla^2 f(x^k)$ is positive definite, then if $\triangle_k \to \infty$, the solution with $\lambda = 0$ is simply the Newton direction. If $\lambda \to \infty$, then $\triangle_k \to 0$, $\|s^k\| \to 0$, and the direction of s^k becomes parallel to the steepest descent direction. Moreover, the trust region direction exists and well defined for sufficiently large λ, even if the Hessian $\nabla^2 f(x^k)$ is not positive definite.

The process is controlled by comparing the predicted reduction in the approximate $|q(s^k) - q(0)|$ and the actual $f(x^k + s^k) - f(x^k)$ reduction in the objective function. The ratio

$$\rho_k = \frac{f(x^k + s^k) - f(x^k)}{q_k(s^k) - q(0)}$$

is calculated. If ρ_k is large enough, then the trust region is expanded in the next iteration; if ρ_k is very small, then the current iteration is repeated with a smaller trust region. Otherwise, the trust region radius remains unchanged.

Trust-Region Algorithm

A simple trust-region algorithm can be described as follows.

Specify a starting point x^0,

trust-region radius Δ_0,

and constants $0 < \mu < \eta < 1,\ 0 < \gamma_1 < 1 < \gamma_2$.

For $k = 0, 1, \cdots$

If a stopping criterion (see page 29) is satisfied, stop.

Solve for s^k

$$\min \quad q_k(s) = f(x^k) + \nabla f(x^k)^T s + \tfrac{1}{2} s^T \nabla^2 f(x^k) s$$

$$\text{s.t.} \quad \|s\| \leq \triangle_k.$$

Compute

$$\rho_k = \frac{f(x^k + s^k) - f(x^k)}{q_k(s^k) - q(0)} = \frac{\text{actual reduction}}{\text{predicted reduction}}.$$

If $\rho_k \leq \mu$ then $x^{k+1} = x^k$ (unsuccessful step).

Else $x^{k+1} = x^k + s^k$ (successful step).

Update Δ_k:

if $\rho_k \leq \mu$, then $\Delta_{k+1} = \gamma_1 \Delta_k$;

if $\mu < \rho_k \leq \eta$ then $\Delta_{k+1} = \Delta_k$;

if $\rho_k \geq \eta$ then $\Delta_{k+1} = \gamma_2 \Delta_k$.

The parameters γ_1 and γ_2 are the trust region radius Δ_k decrease and increase factor, respectively. While updating Δ_k, we usually decrease Δ_k by halving and increase by doubling it. Typical values of η and μ are 0.75 and 0.25, respectively.

The global convergence of the trust region approach can be achieved by finding an approximate solution s^k that lies within the trust region and giving a sufficient reduction in the model. The proof of the global convergence of this algorithm based on nearly exact solutions can be found in Moré and Sorensen [MS83].

2.3 Direct Search Methods

Direct search methods were first suggested in the 1950s and continued to be proposed at a steady rate during the 1960s. The methods proposed at that period were typically motivated by low-dimensional geometric intuition

rather than mathematical theory.

Two important classes of direct search methods are geometry-based and model-based methods. They are described in the following subsections.

2.3.1 Geometry Based Methods

In geometry-based methods, the function values are used to create and maintain a geometric figure, most commonly a simplex, that represents the information known about f at any given iteration. These methods make minimal assumptions about f and do not create a mathematical model of the objective function.

Examples of geometry-based algorithms are the coordinate search method with fixed step sizes [Dav91], the pattern search method based on automata theory [HJ61], the Parallel Direct Search (PDS) method of Dennis and Torczon [DT91]. Variants of simplex-based direct search methods [SHH62] are also geometry-based methods. Usually, these methods need only the function values.

Simplex-based methods are instances of direct search methods. A simplex-based method constructs an evolving pattern of $n + 1$ points in R^n that are

viewed as the vertices of a simplex[2]. A new simplex is formed at each iteration by reflecting the vertex with the largest value of f, over the center of the opposite face of the simplex, or by contracting toward the vertex with the smallest value of f [SHH62]. We discuss the famous Nelder-Mead simplex method in detail in Subsection 2.3.3.

2.3.2 Model Based Methods

Model-based methods use the function values to build a convenient model, such as a quadratic function, through interpolation or approximation. The underlying assumption in defining a model is that $f(x)$ is, in some sense, "nice". Recent survey of such methods is given in [CST97b].

In his thesis [Win69], Winfield proposed to build a quadratic model based on interpolation using available objective function values [Win73]. This model is assumed to be valid in a neighborhood of the current iterate, which is described as a trust region. The model is then minimized within the trust region, hopefully yielding a point with a lower function value. Twenty five years later, Powell [Pow94a] considered a method for constrained optimization in

[2]A simplex is the convex hull of $n+1$ points in general position in R^n, i.e. the (equilateral) triangle for $n = 2$ or tetrahedron for $n = 3$. The volume of the convex hull of the points is positive.

which the objective function and constraints are approximated by linear multivariate interpolation. Later, he described an algorithm [Pow94b] for unconstrained optimization using a multivariate quadratic interpolation model of the objective function in a trust region framework. The main difficulty in practice to use this method is that one must retain the interpolation point set with certain geometric properties at each iteration. This method was explored further recently by Conn and his co-workers [CST97a] [CST97b]. They proposed an approach to handle the geometry of the interpolation set using the so-called Newton fundamental polynomials.

2.3.3 Derivative Free Methods

There are two essential ingredients of derivative free methods. The first is to pick better points. In geometry-based methods, the algorithm should be designed to exploit the next place to sample. In model-based methods, the expectation is that the minimum of the surrogate model will predict suitable points. The second important ingredient is to determine appropriate search subspaces. Different ways to determine such search subspaces result in different algorithms.

Nelder-Mead Simplex Method

The most famous geometry-based direct search method was proposed by Nelder and Mead [NM65]. This method is based on the idea in [SHH62] of creating a sequence of simplices, but deliberately modified, so that the simplex can change shape and thereby "adapt itself to the local landscape" [NM65].

According to [Wri00], at each iteration of the Nelder-Mead algorithm, a current simplex is defined by its $n + 1$ vertices, each point in R^n, along with the corresponding values of f. The "best" vertex corresponds to the lowest function value, with an analogous definition of the worst point. There are five possible operations on the simplex: reflection, expansion, outside contraction, inside contraction and shrinking. A Nelder-Mead iteration has two possible outcomes: (1) a single new point replaces the worst vertex; or (2) if a shrink is performed, the new simplex contains the best point from the previous iteration and n new points closer to the best point than the previous ones. After calculating one or more trial points and evaluating f at these points, each iteration generates a different simplex for the next iteration, where, except for a shrink, the function value at the new vertex is strictly better than the function value at the old worst vertex. A non-shrink Nelder-Mead iteration requires one (for reflection) or two (for an expansion or either

form of contraction) function evaluations, a shrink iteration requires $n + 2$ function evaluations.

A typical iteration of the Nelder-Mead algorithm is outlined by [Wri96] as follows.

1. **Initialization.** Let a starting point x be given.

   Set parameters $\frac{1}{2} \leq \rho \leq 2$, $1 < \chi$, $0 < \gamma < 1$, and $\frac{1}{4} \leq \sigma < 1$ for reflection, expansion, contraction and shrink, respectively.

2. **Order.** Order the $n + 1$ vertices to satisfy $f(x^1) \leq f(x^2) \leq \ldots \leq f(x^{n+1})$, using a consistent tie-breaking rule for the equality cases.

3. **Reflect.** Compute a *reflection point* x^r from

   $$x^r = \bar{x} + \rho(\bar{x} - x^{n+1}),$$

   where $\bar{x}$ is the centroid of the n best vertices (all except x^{n+1}), i.e., $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x^i$, and ρ is the reflection factor.

   Evaluate $f_r = f(x^r)$.

   If $f_1 \leq f_r < f_n$, accept the reflected point x^r, remove x^{n+1} and terminate the iteration. Goto Step 2.

4. **Expand.** If $f_r < f_1$, calculate a *expansion point* x^e from

   $$x^e = \bar{x} + \chi(x^r - \bar{x}),$$

where χ is the expansion factor. Evaluate $f_e = f(x^e)$.

If $f_e < f_r$, accept x^e, remove x^{n+1} and terminate the iteration;

Otherwise (if $f_e \geq f_r$), accept x^r, remove x^{n+1} and terminate the iteration. Goto Step 2.

5. **Contract.** If $f_r \geq f_n$, perform a *contraction* between $\bar{x}$ and the better of x^{n+1} and x^r.

   a. Outside Contraction. If $f_n \leq f_r < f_{n+1}$ (i.e., x^r is strictly better than x^{n+1}), perform an *outside contraction*: calculate

   $$x^{oc} = \bar{x} + \gamma(x^r - \bar{x}),$$

   where γ is the contraction factor, and evaluate $f_{oc} = f(x^{oc})$.

   If $f_{oc} \leq f_r$, accept x^{oc}, remove x^{n+1} and terminate the iteration; Goto Step 2.

   Otherwise, goto step 6 (perform a shrink).

   b. Inside Contraction. If $f_r \geq f_{n+1}$ (i.e., x^{n+1} is better than x^r), perform an *inside contraction*: calculate

   $$x^{ic} = \bar{x} - \gamma(\bar{x} - x^{n+1}),$$

   where γ is the contraction factor, and evaluate $f_{ic} = f(x^{ic})$.

   If $f_{ic} < f_{n+1}$, accept x^{ic}, remove x^{n+1} and terminate the iteration;

Goto Step 2.

Otherwise, goto Step 6 (perform a shrink).

6. **Perform a shrink step.** Define n new vertices by

$$x^i = x^1 + \sigma(x^i - x^1),\ i = 2, \ldots, n+1,$$

where σ is the shrink factor, and evaluate f at these points.

If stopping criteria satisfied, then terminate the algorithm.

Otherwise, terminate the iteration; Goto Step 2.

Despite major efforts, only very weak convergence results [LRWW98] have been established, and only in one and two dimensions, for the original Nelder-Mead method. The convergence theory for a Nelder-Mead variant method, called multi-directional search algorithm, is given by V.J. Torczon [Tor89]. The Nelder-Mead simplex method is widely used due to its simplicity and using only function value decrease requirement. It is also applicable for non-smooth problems, and where the function is not given explicitly.

Derivative Free Optimization (DFO) Algorithm

DFO [CST97b] is a model-based trust region method that exploit, but does not require smoothness in the objective function and attempts to preserve

the convergence properties of their gradient-based counterparts. The idea is to use a smooth quadratic model to approximate the objective function, when the function evaluations are expensive. One then optimizes a much cheaper surrogate model, instead of the function itself, and makes considerable progress in obtaining improved solutions at a moderate cost. The model is assumed to approximate the objective function well in a so-called trust region, typically a ball centered at the current iterate, x^k, of the form

$$B_k = \{x \in R^n \mid \|x - x^k\| \leq \Delta_k\}.$$

The radius of this ball, Δ_k, is called the trust region radius and indicates how far the model is thought to represent well the objective function. A new trial point is then computed, which minimizes the model within the trust region, and the true objective function is evaluated at this point. If the achieved objective function reduction is sufficient, compared with the reduction predicted by the model, the trial point is accepted as the new iterate and the trust region is centered at the new point and possibly enlarged. On the other hand, if the achieved reduction is poor compared with the predicted one, the current iterate is typically unchanged and the trust region is reduced. This process is repeated until convergence occurs.

The main steps of a typical trust region method can be described as

follows:

1. Given a current iterate build a good local approximation model.

2. Choose a neighborhood around the current iterate where the model is "trusted" to be accurate. Minimize the model in this neighborhood.

3. Determine if the step is successful by evaluating the true function at the new point and comparing the true reduction in value of the objective with the reduction predicted by the model.

4. If the step is successful, accept the new point as the next iterate. Increase the size of the trust region, if the success is really significant. Otherwise, reject the new point and reduce the size of the trust region.

5. Repeat until convergence.

It has been shown [Pow94b] [CST97b] that quadratic interpolation can be applied successfully in combination with a trust region method.

Quadratic Interpolation Consider the problem of interpolating a given function $f(x),\ x \in R^n$ by a quadratic polynomial $Q(x)$ at a chosen set of interpolation points, $X = \{x^j \mid x^j \in R^n, j = 1, \cdots, p\}$. Suppose that $\{\phi_i(\cdot)\}_{i=1}^q$ is a basis in the space of quadratic polynomials. Then, any quadratic polynomial $Q(x) = \sum_{i=1}^q \xi_i \phi_i(x)$ can be represented in terms of these basis functions

with some coefficient vector $\xi = (\xi_1, \ldots, \xi_q)^T$. To determine a full set of the quadratic model coefficients ξ, we needs

$$q = 1 + n + \frac{1}{2}n(n+1) = \frac{1}{2}(n+1)(n+2)$$

parameters.

The interpolation coefficients can be obtained by solving the system of linear equations:

$$\sum_{i=1}^{q} \xi_i \phi_i(x^j) = f(x^j) \quad j = 1, \ldots, p. \tag{2.4}$$

The coefficient matrix of this system is

$$\Phi(X) = \begin{pmatrix} \phi_1(x^1) & \cdots & \phi_q(x^1) \\ \vdots & & \vdots \\ \phi_1(x^p) & \cdots & \phi_q(x^p) \end{pmatrix}. \tag{2.5}$$

In order to have a unique solution of this system, we need the coefficient matrix $\Phi(X)$ to be square, i.e., $p = q$ and $\Phi(X)$ to be nonsingular. Numerical solvability of the equation system depends on the conditioning of the matrix $\Phi(X)$. If we have a full basis of polynomials, then we have $p = q = \frac{1}{2}(n+1)(n+2)$.

Geometric Condition A set of points X is called *poised*, with respect to a given subspace of polynomials, if at the points in X the function $f(x)$

can be interpolated uniquely by polynomials from this subspace, i.e., a set of points X is poised if $\Phi(X)$ is nonsingular.

A set of points X is called *well-poised*, if it remains poised under small perturbations. For example, if $n = 2$, six points almost on a line may make a poised set. However, since some small perturbation of the points might make them aligned, it is not a well-poised set. Indeed, the coefficient matrix $\Phi(X)$ can be very ill-conditioned, and the interpolation polynomial is likely to provide a very bad approximation of the function.

As we mentioned, for a given set of points X, a set of function values and given set of basis functions, an interpolation polynomial (from a given space of polynomials) exists and is unique if and only if $\Phi(X)$ is square and nonsingular. In other words, the set X is poised if the determinant

$$\delta(X) = \det \begin{pmatrix} \phi_1(x^1) & \cdots & \phi_p(x^1) \\ \vdots & & \vdots \\ \phi_1(x^p) & \cdots & \phi_p(x^p) \end{pmatrix} \tag{2.6}$$

is nonzero. If the condition number of $\Phi(X)$ is small, then system (2.4) remains stable under small perturbations, thus the set X is well poised.

Newton Fundamental Polynomial The most difficult step in the DFO algorithm is updating the interpolation set. This is so, because depending on

which point of the set is removed, the conditioning of the coefficient matrix may deteriorate (the matrix can become singular or ill conditioned), making the next iteration impossible or more difficult. A useful tool to choose a good interpolation set or to create a new point that improves the poisedness of the coefficient matrix is provided by the Newton Fundamental Polynomials (NFP) basis.

For a given set of interpolation points X, the basis of NFP is such that $\Phi(X)$ has the structure shown in Figure 2.3. The values in the white space can be arbitrary real numbers.

The set of interpolation points is partitioned into three subsets (blocks) X^0, X^1 and X^2 which correspond to the constant term, linear terms and quadratic polynomials, respectively. Hence, X^0 has a single element, X^1 has n elements and X^2 has $n(n+1)/2$ elements. The basis of NFP $N_i(\cdot)$ is also partitioned into three blocks $N_i^0(\cdot)$, $N_i^1(\cdot)$ and $N_i^2(\cdot)$ with the appropriate number of elements in each block. The unique element of $N_i^0(\cdot)$ is a polynomial of degree zero, each of the n elements of $N_i^1(\cdot)$ is a polynomial of degree one and each of the $n(n+1)/2$ elements of $N_i^0(\cdot)$ is a polynomial of degree two.

The basis elements and the interpolation points are in one-to-one cor-

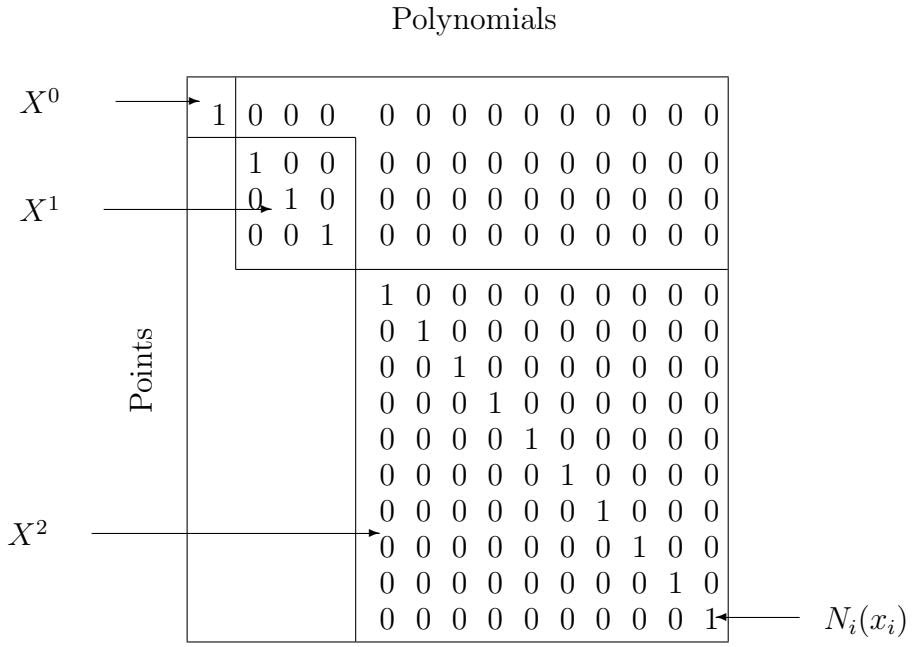

Figure 2.3: Structure of the coefficient vectors of the NFP basis

respondence, so that points from block X^l correspond to polynomials from block $N_i^l(\cdot)$. A Newton polynomial $N_i(\cdot)$ and a point X^i correspond if and only if the value of that polynomial at that point is 1 and its value at any other point in the same block or in any prior blocks is 0. In other words, if X^i corresponds to N_i, then $N_i(x^i) = 1$, and $N_i(x^j) = 0$ for all index j within the first l blocks (see 2.3).

More details on this and multivariate interpolation in general can be found in [SX95]. For instance, if we consider quadratic interpolation on a regular grid in the 2D-plane, we require six interpolation points using three blocks

$$X^{[0]} = \{(0,0)\},\ X^{[1]} = \{(1,0),(0,1)\},\ X^{[2]} = \{(2,0),(1,1),(0,2)\},$$

corresponding to the basis functions $1, x_1, x_2, x_1^2, x_1x_2$ and x_2^2 respectively. Applying the procedure of constructing the basis of NFP as described in [SX95] we obtain

$N_1^{[0]} = 1, N_1^{[1]} = x_1, N_2^{[1]} = x_2,$

$N_1^{[2]} = \frac{1}{2}(x_1^2 - x_1), N_2^{[2]} = x_1x_2, N_3^{[2]} = \frac{1}{2}(x_2^2 - x_2).$

The outline of a derivative free trust region algorithm can be found in [CST97b] and [Sch00]. Our implementation following [Sch00] with slight modification is described below.

step 0: Initialization

Let x^s be a starting point, and the value $f(x^s)$ be given. Choose an initial trust region radius $\triangle_0 > 0$. Choose at least n additional points[3] not further than $\triangle_0$ away from x^s to create an initial well-poised interpolation set X and initial basis of NFP. Determine $x^0 \in X$ which has the best objective function value; i.e., $f(x^0) = \min_{x^i \in X} f(x^i)$.

Set $k = 0$, parameters η, η_0 and η_1 to measure progress: $0 < \eta = \eta_0 < \eta_1 < 1$, ε for stopping, γ_0, γ_1 and γ_2 for the trust region expansion and contraction.

step 1: Build the model

Using the interpolation set X and the basis of NFP, build the interpolation model $Q_k(x)$.

step 2: Minimize the model within the trust region

Set $B_k = \{x : \|x - x^k\| \leq \triangle_k\}$. Compute the point $\hat{x}^k$ such that

$$Q_k(\hat{x}^k) = \min_{x \in B_k} Q_k(x).$$

Compute $f(\hat{x}^k)$ and the ratio

$$\rho_k \equiv \frac{f(x^k) - f(\hat{x}^k)}{Q(x^k) - Q(\hat{x}^k)}.$$

[3] In [Sch00], it is "one additional point". They build up the model from a few points while we start with a full linear model in our implementation.

step 3: Update the interpolation set

- If $\rho_k \geq \eta_0$, include $\hat{x}^k$ in X, dropping one of the existing interpolation points.
- If $\rho_k < \eta_0$, include $\hat{x}^k$ in X, if it improves the quality of the model.
- If $\rho_k < \eta_0$ and there are less than $n+1$ points in the intersection of X and B_k, generate a new interpolation point in B_k, while preserving /improving well-poisedness.
- Update the basis of the NFP.

step 4: Update the trust region radius

- If $\rho_k \geq \eta_1$, increase the trust region radius

$$\triangle_{k+1} \in [\triangle_k, \gamma_2 \triangle_k].$$

- If $\rho_k < \eta_0$ and the cardinality of $X \cap B_k$ was greater[4] than $n+1$ when $\hat{x}^k$ was computed, reduce the trust region

$$\triangle_{k+1} \in [\gamma_0 \triangle_k, \gamma_1 \triangle_k].$$

- Otherwise, set $\triangle_{k+1} = \triangle_k$.

[4] In [Sch00], it is "less".

step 5: Update the current iterate

Determine $\bar{x}^k$ with the best objective function value

$$f(\bar{x}^k) = \min_{x^i \in X, x^i \neq x^k} f(x^i).$$

If improvement is sufficient (w.r.t. predicted improvement)

$$\bar{\rho}_k \equiv \frac{f(x^k) - f(\bar{x}^k)}{Q(x^k) - Q(\hat{x}^k)} \geq \eta.$$

Set $x^{k+1} = \bar{x}^k$. Increase k by one and go to step 1.

This outline provides only a framework. Practical algorithms involve a number of additional features such that stopping criteria, choosing a point to drop, the criteria of improvement in the model. We will discuss those techniques and the implementation issues in Chapter 4.

It has been proved that DFO is globally convergent to a local minimum, provided that the approximation model is at least fully linear to ensure a reasonable approximate result of the objective function [CST97a]. It has been reported that in practice DFO finds the global minimum (or a "good" local minimum) in many cases [CST97a].

Chapter 3

Algorithms for Solving the LJ Cluster Problem

The idea of our approach is that, using direct search methods such as Nelder-Mead Simplex and DFO methods for global search in the search space, to follow by local search from the resulting best point or from the best point at each iteration, if the best point is improved. We use the Quasi-Newton method with BFGS update as the local search procedure.

3.1 The Nelder-Mead Based Approach

We combine the Nelder-Mead method with local search in three ways.

3.1.1 Approach I: General Combination Approach

Our first approach to the problem of globally minimizing the LJ potential function is combining the Nelder-Mead method and the local search method. The approach can be described as follows.

1. Generate randomly N points in 3D, each point representing an atom.
2. Minimize the LJ potential function using the Nelder-Mead method.
3. Perform local search started from the result obtained from the Nelder-Mead method.

We applied this procedure to problems with number of atoms $N = 2, 3, \ldots, 10$. The results (in Table 5.1, Chapter 5) show that this random sampling procedure can only detect the global optimum for $N \leq 5$. Therefore, it seems necessary to modify this procedure in order to be able to solve larger problems.

3.1.2 Approach II: Build up Combination Approach

A simple idea is to exploit the special structure of the LJ potential function and modify the search mechanism accordingly. Looking at the form of the LJ function, we notice that good solutions should possess some or all of the following characteristics:

1. Atoms should not be too close to each other.

2. Atoms should not be too far away from each other.

3. The distance between all pairs of atoms should be close to 1.0, since at 1.0, the LJ pair potential attains its minimum.

According to these simple observations, it is possible to substitute the initial random generation of points and a single pass Nelder-Mead algorithm by a generation mechanism, which tends to favor point configurations possessing the above characteristics. As a first attempt in this direction, we substituted the random generation procedure with the following procedure.

1. Start with two points with distance one.

2. Add one randomly generated point.

    - Minimize the LJ potential function using the Nelder-Mead method.

- Perform local search started from the result obtained from the Nelder-Mead simplex method;
- Keep track of the best point obtained so far.

3. If the number of the points is less than the number of points required, go to step 2.

   Otherwise, report the solution and stop.

With this improved procedure, we successfully located the best known solution for micro-cluster of N = 2,3,4,5,7,8,9,11,14,15,17,20 and 25. The results are given in Table 5.2 to 5.4 in Chapter 5. It is worth noting that the cluster of $N = 6$ has been detected by putting 6 points on the vertices of a regular octahedron and starting minimization with this configuration. However, in general, to find the regular polyhedron is as difficult as the original problem itself. We tried to modify step 2 of the procedure by adding one point from inside and outside of the convex hull (simplex). Unfortunately, this heuristic gave no effect on the results.

3.1.3 Approach III: Build up Incorporation Approach

In this approach, instead of starting a local search with the resulting best point obtained from the Nelder-Mead method, we incorporate a local search in the iterates of the Nelder-Mead simplex method. At each iteration of the Nelder-Mead simplex method, whenever a better point is determined, we start a local search from this point. The result is compared with the one of the previous iteration. We keep only the best one. This procedure can be described as follows:

1. Start with two points with distance one.

2. Add one randomly generated point.

   - Minimize the LJ potential function using the Nelder-Mead method;
   - Start local search at the best point of each iteration;
   - Keep track of the best point obtained so far.

3. If the number of points is less than the number of points required, go to step 2.

   Otherwise, report the solution and stop.

Using the build up incorporation approach, we located the best known solution for micro-clusters for all $N \leq 30$. The results are given in Tables 5.2 to 5.4 in Chapter 5.

3.2 The DFO Based Approach

The results of the previous Nelder-Mead simplex method based approaches show that the geometry based direct search methods can be used to locate the global optimum of the LJ potential very well for micro-clusters of $N \leq 30$. Comparing the results of the three approaches, the third one gives the best results. The DFO method is a model based direct search method that differs from Nelder-Mead simplex method. Naturally, our DFO based method can be best described as "build up incorporation approach based on DFO" approach. It is given as follows.

Approach IV: Build up Incorporation Approach Based on DFO

1. Start with two points with distance one.

2. Add one randomly generated point.

   - Minimize the LJ potential function using the DFO method.
   - Whenever the trust region center moves to a new point, start the local search from that new center.
   - Keep track of the best solution obtained so far.

3. If the number of points is less than the number of points required, go to step 2.

   Otherwise, report the solution and stop.

3.3 Local Search Procedures

Both implemented direct search methods are furnished with a local search descent algorithm. The Quasi-Newton method with BFGS update, which we have used in our algorithms, is given in Subsection 2.2.1.

Chapter 4

Implementation

In this chapter, we first focus on the implementation of the direct search algorithms: Nelder-Mead and DFO. Later, we give the build up scheme for a given atomic cluster, the local search procedures and the method of visualizing the structure of a cluster at its global minima.

All the procedures were implemented in MATLAB. We provide the function evaluation, gradient and Hessian computing procedures of the LJ cluster problem. The computational test was performed on an Intel Pentium III 677MHz RAM 128MB PC and IBM RS6000 RAM 1GB workstation. The time is not measured with this implementation in MATLAB. The Virtual Reality Modelling Language (VRML 1.2) was used as the 3D geometry analysis

tool.

4.1 Implementation of the Nelder-Mead Simplex Method

The implementation of the Nelder-Mead simplex method is quite straightforward. Figure 4.1 gives the structure of the method[1]. Figure 4.2 is the flowchart specifying how to generate a new point at each iteration.

4.1.1 Starting Points

Given a point `xin` of dimension n, we discuss two ways how to build a simplex of $n + 1$ points:

1. Shifting the unit simplex of dimension n from the origin to `xin`.

```
u = eye(n);
v(:,1) = xin;
for j=1:n
    y = u(:,j)+x;
    v(:,j+1) = y;
```

[1]The rectangle with round corner in the flowcharts represents a sub-procedure.

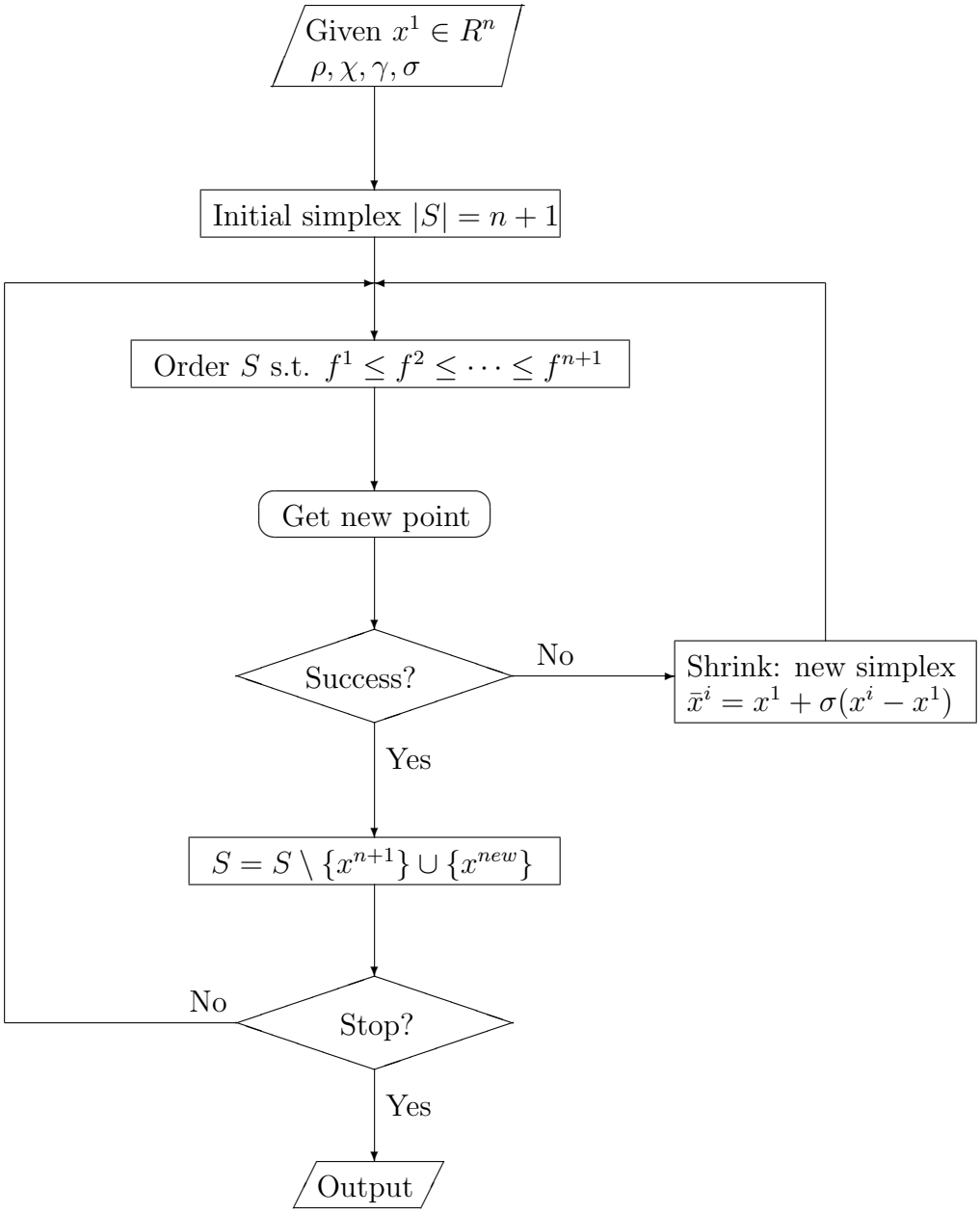

Figure 4.1: Structure of the Nelder-Mead Simplex method.

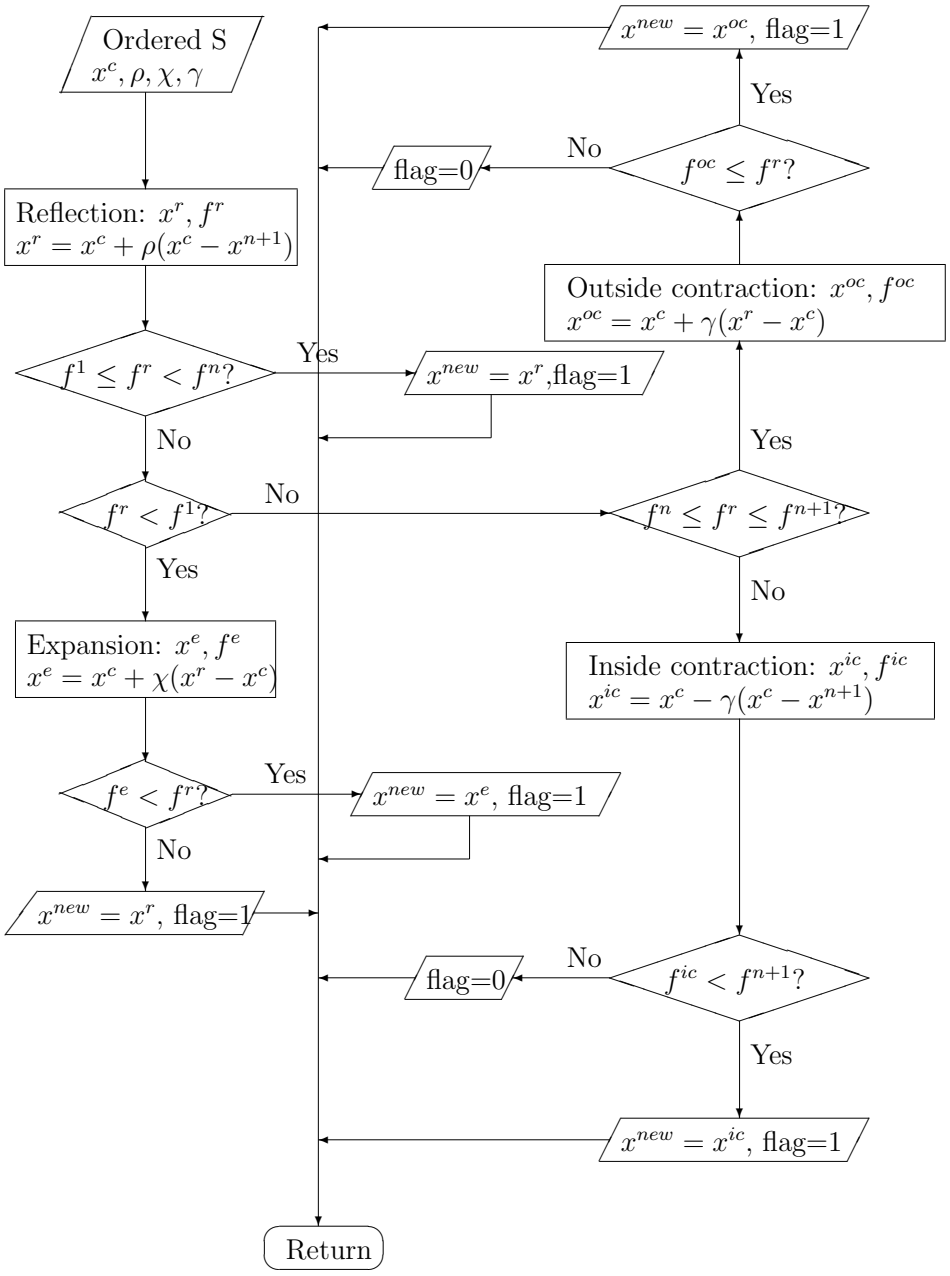

Figure 4.2: Structure of the "Get new point" subroutine.

```
        x(:) = y;
        f = feval(funfcn,x,varargin{:});
        fv(1,j+1) = f;
    end
```

2. Using a modified point set suggested by L.Pfeffer at Stanford [MAT]

```
usual_delta = 0.05; zero_term_delta = 0.0075;
for j = 1:n
    y = xin;
    if y(j) ~= 0
       y(j) = (1 + usual_delta)*y(j);
    else
       y(j) = zero_term_delta;
    end
    v(:,j+1) = y;
    x(:) = y; f = feval(funfcn,x,varargin{:});
    fv(1,j+1) = f;
end
```

where v is a matrix of n-by-$(n+1)$ which stores the coordinates of $n+1$

points, and vector `fv` stores the corresponding $n + 1$ function values.

The first method shifts the unit simplex of dimension of n from the origin to the initial point. This way guarantees that the simplex has a "good geometry". The second method creates a simplex with specified edge length and orientation that depends on the given starting point of the coordinate directions. If it is in the same direction as an optimal solution, this initial simplex may push the process fast towards the optimum. The risk is that the constructed simplex may be very flat. In other words, the shape of the initial simplex is constructed with sharp angles. If the coordinate direction of the given starting point is orthogonal to the direction towards an optimal solution, then it will take more iterations to find an optimal solution or, in the worst case, fail to find an optimal solution at all. The parameters `usual_delta` and `zero_term_delta` are used to adjust the simplex orientation and can be modified as needed.

4.1.2 Termination

The algorithm terminates based on two criteria: either the function values at the vertices are too close, or the simplex has become very small. In practice, we stop the procedure if the following condition is satisfied:

```
max(max(abs(v(:,2:n+1)-v(:,1)))) <= tolx
& max(abs(f(1)-f(2:n+1))) <= tolf
```

where `tolx` and `tolf` are tolerance for the vertices and the function values, respectively. `v(:, 1:n+1)` contains the coordinates of the $n+1$ points.

4.1.3 Iteration

At each iteration of the Nelder-Mead algorithm, the current simplex is defined by its $n+1$ vertices, each a point in R^n, along with the corresponding values of f. The "best" vertex corresponds to the lowest function value, with an analogous definition of the worst point.

In the implementation, we first evaluate the objective function at $n+1$ points of the initial simplex and sort them. Then, we define the centroid point `xbar` of the n best points

```
xbar = sum(v(:,1:n),2)/n;
```

and the four points correspond to four possible operations on the simplex, reflection(`xr`), expansion(`xe`), outside

contraction(`xoc`), inside contraction(`xic`).

At each iteration, we try to obtain one new point that satisfies the function value improving requirement, from one of the four operations as it is

described in the algorithm and illustrated by the flowchart of the "Get new point" sub-procedure in Figure 4.2. If such a point is found, then we use it to replace the worst vertex in the current simplex. Otherwise, a shrink operation is performed.

The shrink operation produces a new simplex which contains the best point from the previous iteration and n new points.

We repeat this process until the termination condition is satisfied.

4.2 Implementation of the DFO Algorithm

The implementation of the DFO algorithm is much more complicated. In our DFO implementation, we distinguish two working spaces. One is the real problem space, in which we evaluate the objective function value and then evaluate the points generated in the model space. Another is the model space, in which we perform the search by minimizing the quadratic model over a trust region, centered at a "good" point. The "goodness" of a point is measured by its corresponding function value, and the geometry property of the interpolating point set that it is included.

4.2.1 The Implementation

The logical structure of our DFO implementation is illustrated in Figure 4.3. Based on this logical structure, a main procedure called DFO and thirteen sub-procedures are created. They are described as follows.

Main procedure DFO

The procedure DFO is declared with the following form:

```
function [x,fval,exitflag,output]=dfo(funfcn,x,options,varargin)
```

DFO calls a user provided subroutine `funfcn` that evaluates the objective function, a given initial point `x` and a set of `options` as input. The `options` can be set up as follows.

```
options =
  optimset('display','final','maxiter',2000,'maxfunevals',10000);
```

If this part is not set up by the user, DFO provides a default setting as above. The `varargin` is a MATLAB input argument list. It allows any number of arguments to the function `funfcn`.

DFO returns the solution `x`, function value `fval`, a flag `exitflag`, indicating the status, when the program terminates, and a structure `output`

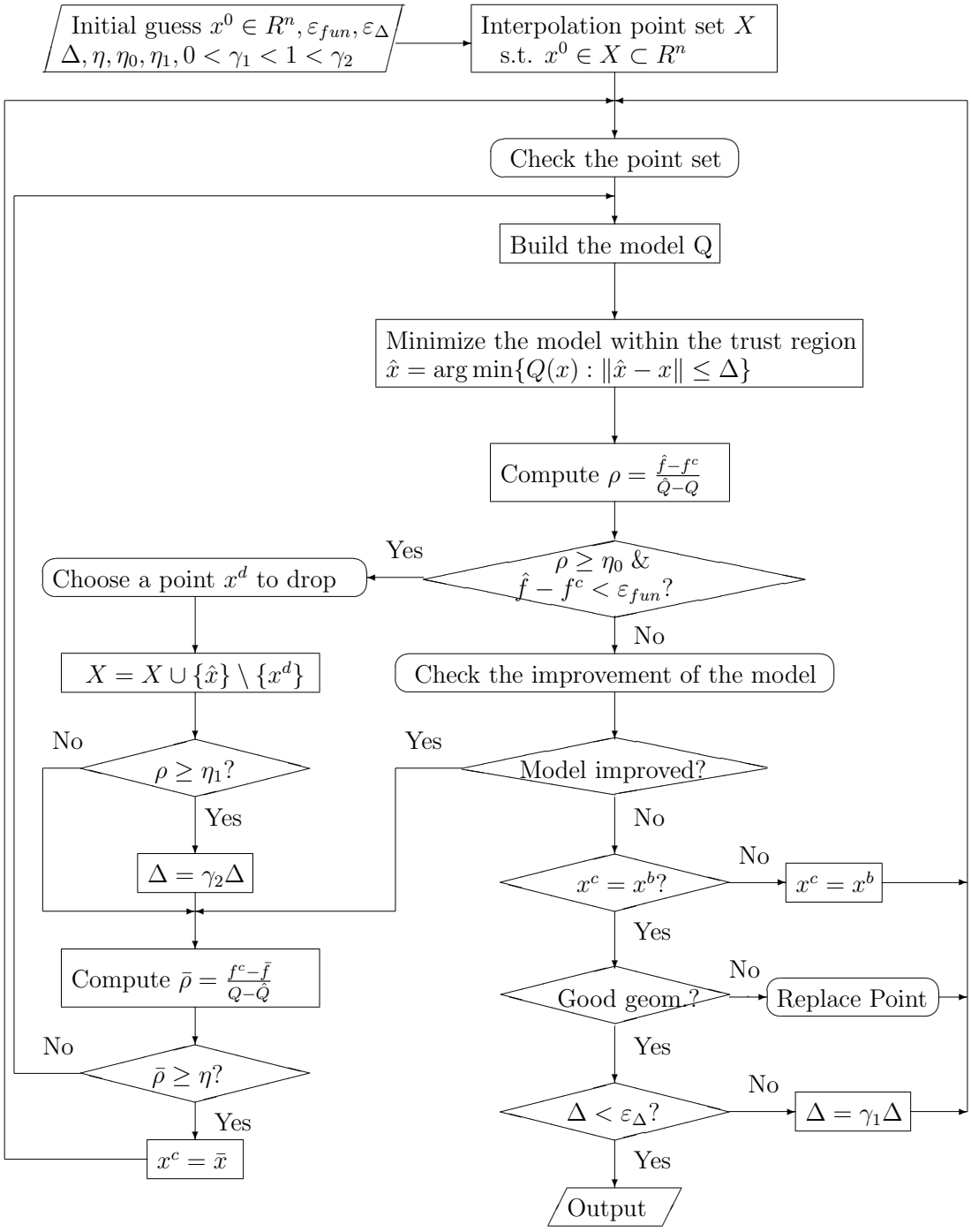

Figure 4.3: Structure of the DFO algorithm.

with the number of iterations, the number of function evaluations and the name of the algorithm.

At the beginning of the algorithm, DFO reads a set of parameter values. This part can be modified by the user to meet his or her needs. The default parameter values and their meaning are as follows.

```
epsTrust = 1e-3;  % Minimum value of the trust region radius.
epsDet = 1e-12;   % Determinant tolerance.
delta0 = 1;       % Initial trust region radius.
eta0 = 0.45;      % Parameter.
eta = eta0;       % Parameter.
eta1 = 0.75;      % Parameter.
gama1 = 0.5;      % Trust region radius decreasing factor.
gama2 = 2;        % Trust region radius increasing factor.
epsDist = 0.5;    % Minimum distance allowed between 2 points.
epsFun = 1e-8;    % Function reduction tolerance.
```

The main loop is safeguarded by the number of the function evaluations and the number of the iterates. The stopping criteria is checked inside the main loop. Three flags are used to control the data flows. They are described as follows.

flagdelta: The flag `flagdelta` indicates the change of the trust region radius delta.

```
flagdelta = 0, % delta is not changed;
flagdelta = 1, % increase delta;
flagdelta = 2, % decrease delta.
```

flaghat: The flag `flaghat` indicates the acceptance of the point `xhat`. The point `xhat` is accepted if either it gives a good reduction of the function value or it improves the model.

```
flaghat = 0, % xhat is rejected;
flaghat = 1, % xhat is accepted by the means of the function
             % reduction.
flaghat = 2, % xhat is accepted by the means of the model
               improvement.
```

flagcheck: The flag `flagcheck` indicates if the new point `xcheck`, is added into the interpolation point set to satisfy the minimum linearity model requirement.

```
flagcheck = 0, % there is no point added;
```

```
flagcheck = 1, % there is a point xcheck added.
```

In the next subsections, we'll give the implementation details of the subroutines used in DFO.

Starting points

The initial point set is computed by procedure `setY()`. We take the vertices and the mid-points of the edges of a regular simplex in R^n as an initial point set. The edges of the regular simplex have length $\kappa\Delta_0$, where κ is a scaling factor, Δ_0 is the initial trust region radius. The default value for κ was set to $\frac{\sqrt{2}}{2}$. A corresponding basis of NFP is built up by calling the procedure `setNP()`.

Termination

The algorithm is terminated when the following three criteria are satisfied.

1. The radius of the trust region is small enough, s.t.

$$\Delta \leq \texttt{epsTrust}.$$

2. The final interpolation point set has the "good geometry" property.

3. The center of the current trust region is the point with the best function value.

The default minimum value of the trust region radius is set up to 10^{-3}. The "good geometry" property is interpreted as follows.

1. At least $n + 1$ points are in the trust region.

2. The distance of each point to the current center point is less than two times of the current trust region radius and the interpolation set is well poised.

If stopping criteria 1 and 3 are satisfied, but some of the points are not in the ball of two times of the current trust region radius, we replace the furthest point by a new point generated in the trust region. So the new point guarantees the model improvement in the sense that it maximizes $\delta(X)$ that is defined by (2.6). If criteria 1 and 2 are satisfied, we move the center to the best point. If criteria 2 and 3 are satisfied, the trust region radius is reduced.

The procedure `stop()` is implemented to check these termination criteria.

Trust region minimization (maximization)

On each iteration, the DFO algorithm solves at least one trust region minimization problem of the model function $Q(x)$. Two subroutines are used to solve the constrained inner trust region sub-problems:

$$\min \ \{\nabla Q(x)^T s + \frac{1}{2} s^T \nabla^2 Q(x) s : \|s\| \leq \triangle\}. \tag{4.1}$$

The first one is the MATLAB subroutine `trust()`, in which full eigen value decomposition, based on the secular equation [GL89] [Ste98],

$$\frac{1}{\Delta} - \frac{1}{\|s\|} = 0,$$

is used. The other one is `lmlib()`, using the method of the Levenberg-Marquardt algorithm with the technique of More [Fle87] [Gay81]. This subroutine is implemented in C with special techniques for higher efficiency [Fin96] by Lukas Finschi in Institut fuer Operations Research, Technical University, Zuerich. This `lmlib()` subroutine is integrated in MATLAB using MATLAB API mex.

Dropping a point

The procedure `droptr()` is implemented to choose a point to drop, whenever a new point is obtained that has a better function value within the

minimization step.

In general, we choose a point to drop if this point is too far away from the current trust region or the objective function value at this point is too bad.

Let x^c be the current trust region center point, x^f be the furthest point from x^c and x^w be the point with the worst function value in the interpolation point set. Then we choose a point to drop by applying the following criteria:

if $\|x^f - x^c\| > 4\triangle$ **or** $(\|x^f - x^c\| > \triangle$ **and** $f(x^f) > \frac{1}{N}\sum_{i=1}^{N} f(x^i))$

then drop $= x^f$;

else drop $= x^w$.

We replace the chosen point to drop by the new point. If the resulting interpolation point set has a very "bad" geometry property, then we look for the point closest to the new point x^t. This point x^t therefore becomes an obvious candidate point to drop.

Model improvement

The "goodness" of the model is measured by the determinant $\delta(X)$ defined in (2.6). Let `layer1` < `layer2` < `layer3` be the factors of the improvement

in $\delta(X)$. We try to replace x^f, the point furthest away from the current trust region center by a new point. If such a point is in the trust region, we request that the factor to qualify the improvement in $\delta(X)$ is greater than `layer2`. On the other hand, if x^f is further away from than `layer3` times of the trust region radius Δ, we request the point set remains well poised. Finally, if $\|x^f - x^c\|$ is between `layer1` and `layer3` times the trust region radius Δ, we request the factor of the improvement to be greater than `layer1`. If x^f fails to satisfy the model improvement criteria, we move to the next furthest point and repeat the test. This procedure can be described as follows:

For $i = 1, \cdots$, number of points in a set,

1. Find the furthest point x^f from the current point x^c.

2. Replace x^f by a candidate point x.

3. Calculate

$$\text{ratio} = \frac{\text{determinant after replacement}}{\text{determinant before replacement}}.$$

If $\|x^f - x^c\| \leq \triangle$ and ratio $\geq$ `layer2`, then accept x, return.

Else, if $\|x^f - x^c\| \leq$ `layer3` $\triangle$ and ratio $>$ `layer1`, then accept x, return.

Else, if $\|x^f - x^c\| \geq$ `layer3` $\triangle$ and $\delta(X) > \varepsilon$, then accept x, return.

Else find the next furthest point $x^{f`}$ and goto step 2.

In our implementation, the factor `layer1`, `layer2` and `layer3` are set to 1, 2 and 4, respectively. They can be modified as needed.

The procedure `improvMod()` is implemented to check if including a point improves the quality of a model or not within the minimization step.

Check model

The procedure `checkMod()` is implemented to check if there are $n+1$ points in the current trust region. If there are less than $n+1$ points, we replace the point furthest away from the current trust region center x^f by $\check{x}$ s.t.

$$\check{x} = \arg\max\{\det(\Phi(\bar{X})) : \bar{X} = X \setminus \{x^f\} \cup \{x\}, \|x - x^c\| \leq \Delta\},$$

which is a point on the boundary of the current trust region. The procedure `inTrust()` is used to compute the number of the points in the current trust region. The procedures `maxDist()` and `evalDet()` are used to compute x^f and to evaluate $\det(\Phi(X))$, respectively. They are described in the following subsection.

Other procedures

The procedure `evalDet()` evaluates determinant along a given row index.

The procedure `inTrust()` gets the distance information of the points in the interpolation set with respect to the current center point and the number of points in the current trust-region.

The procedure `maxDist()` finds the farthest point in the interpolation set with respect to the current center point and its index in the model.

The procedure `quad2quad()` transforms the model from the Newton fundamental polynomial form into the standard quadratic $\kappa + g^T x + x^T hx$ function form, where κ is a constant, g is the gradient, h is the Hessian.

The procedure `quadEval()` evaluates the quadratic approximation function value, gradient and Hessian at a given point x.

The procedure `y2quadX()` transforms a point of dimension n into a point of dimension $(n+1)(n+2)/2$.

The procedure `message()` handles the information to print on the screen to user.

4.2.2 The Test Problems

To test our implementation, we used standard problems from the optimization literature [SOP]. The size of the problems ranges from 2 to 30 dimen-

sions. We count the number of the actual function evaluations and the number of the iterations required to return a solution with a specified tolerance level.

Our tests includes Rosenbrock's "banana" function, Powell's quadratic function, and the non-linear functions s350, s351, s370, s371 and s391. They are of dimensions 2, 4, 4, 4, 6, 9 and 30, respectively. They are all minimization problems. These problems are given as follows.

1. Rosenbrock's "banana" function

```
n = 2
f = 100*((x(2)-x(1))^2)^2+(1-x(1))^2
```

2. Powell's quadratic function

```
n = 4
f = (x(1)+10*x(2))^2+5*(x(3)-x(4))^2+(x(2)-2*x(3))^4+
    10*(x(1)-x(4))^4;
```

3. s350

```
n = 4;
```

```
f: sum {i in 1..11} (y[i] -
  x[1]*(u[i]^2+x[2]*u[i])/(u[i]^2+x[3]*u[i]+x[4]))^2;
where
        y:=     u:=
  1   0.1957  4.0000
  2   0.1947  2.0000
  3   0.1735  1.0000
  4   0.1600  0.5000
  5   0.0844  0.2500
  6   0.0627  0.1670
  7   0.0456  0.1250
  8   0.0342  0.1000
  9   0.0323  0.0833
  10  0.0235  0.0714
  11  0.0246  0.0625;
```

4. s351

```
n = 4;
f: 10^4*sum{i in 1..7}
```

```
  (((x[1]^2+x[2]^2*a[i]+x[3]^2*a[i]^2)/(1+x[4]^2*a[i])
   -b[i])/b[i])^2;
where
     a:=          b:=
 1   0.0         7.391
 2   0.00043     11.18
 3   0.00100     16.44
 4   0.00161     16.20
 5   0.00209     22.20
 6   0.00348     24.02
 7   0.00525     31.32;
```

5. s370

```
n = 6;
f: x[1]^2 + (x[2] - x[1]^2 - 1)^2 +
  sum {i in 2..30}(sum{j in 2..6}
  (j-1)*x[j]*((i-1)/29)^(j-2) -
  (sum {j in 1..6}x[j]*((i-1)/29)^(j-1))^2 - 1)^2;
```

6. s371

```
n = 9;
f: x[1]^2 + (x[2] - x[1]^2 - 1)^2 +
  sum {i in 2..30}(sum{j in 2..9}
  (j-1)*x[j]*((i-1)/29)^(j-2) -
  (sum {j in 1..9}x[j]*((i-1)/29)^(j-1))^2 - 1)^2;
```

7. s391

```
n = 30;
v[i in 1..N, j in 1..N] = sqrt (x[i]^2 +i/j);
alpha{i in 1..N} = 420*x[i] + (i-15)^3 +
    sum {j in 1..N} v[i,j]*((sin(log(v[i,j])))^5 +
    (cos(log(v[i,j])))^5);
f: sum {i in 1..30} alpha[i];
```

4.2.3 Numerical Results and Discussion

We report some of the results we obtained during implementing and testing our implementation of the DFO algorithm. In all the tests, the parameters were set to the default values, unless specified else. The results about the Rosenbrock function are compared with the ones reported by

Powell [Pow96]. We made also a comparison between the two subroutines `trust()` and `lmlib()` that we used to solve the trust region subproblems in our DFO.

In the following tables, ε_Δ denotes the minimum value of the trust region radius. #iter denotes the number of the iterations. #fun denotes the number of the function evaluations. x^* denotes the solution and $f(x^*)$ denotes the objective function value at the solution x^*.

We first tested our DFO on Rosenbrock's function. The comparisons reported in Table 4.2 are based on the results reported in [Pow96]. The numbers of the iterations in Table 4.2 are not available. The initial trust region radius was set to $\Delta_0 = 0.1$. The starting point was set to $x^0 = (-1.2, 1)$. The results are given in Table 4.1 for DFO:trust and 4.3 for DFO:lmlib.

It is easy to see that our DFO requires much less function evaluations than Powell's for the same trust region tolerance. Note that in Table 4.1, when ε_Δ is set to 1/80 requires 3 more function evaluations than when ε_Δ is set to 1/160. Because our DFO is terminated when all the three conditions described in subsection 4.2.1 are satisfied, the former reaches the trust region tolerance level earlier, but it requires more function evaluations to satisfy other conditions. So these are normal situations.

ε_Δ	#iter	#fun	x^*	$f(x^*)$
1/10	15	31	(0.7089, 0.5672)	0.1251
1/20	18	37	(0.7385, 0.6075)	0.0978
1/40	33	60	(1.0006, 1.0011)	3.4442e-007
1/80	37	67	(1.0006, 1.0011)	3.4442e-007
1/160	37	64	(1.0000, 0.9983)	8.5312e-010
1/640	41	71	(1.0000, 0.9983)	8.5312e-010

Table 4.1: DFO:trust on Rosenbrock's function

ε_Δ	#iter	#fun	x^*	$f(x^*)$
1/10		71	(0.849, 0.724)	2.4e-2
1/20		97	(0.967, 0.938)	1.6e-3
1/40		111	(1.001, 1.001)	1.7e-7
1/80		118	(1.001, 1.001)	1.7e-7
1/160		124	(1.000, 1.000)	4.9e-9

Table 4.2: Results on Rosenbrock's function reported by Powell

As we can see, in Table 4.1, when ε_Δ is set to larger or equal to 1/20, DFO failed on Rosenbrock's function to converge to the optimal solution. Figure 4.4 illustrates the movement of the trust region center for ε_Δ 1/20 and 1/40,

ε_Δ	#iter	#fun	x^*	$f(x^*)$
1/10	32	51	(0.9937, 1.0029)	4.0356e-005
1/20	32	53	(0.9937, 1.0029)	4.0356e-005
1/40	33	55	(0.9937, 1.0029)	4.0356e-005
1/80	34	56	(1.0009, 1.0032)	8.6922e-007
1/160	37	61	(0.9999, 0.9986)	2.1958e-008
1/640	51	67	(1.0000, 0.9994)	1.7481e-011

Table 4.3: DFO:lmlib on Rosenbrock's function

respectively. It can be seen that the parameter ε_Δ, the trust region radius threshold, may affect the results. If ε_Δ is set to a large value, the algorithm may terminates inappropriately earlier.

The computational results with Powell's function are given in Tables 4.4 and 4.5. We take a initial point $x^0 = (1, -2, 0, 3)$. The initial trust region radius is set to 1. As we can see, for the same ε_Δ level, DFO:trust requests less function evaluations, but DFO:lmlib gives better solutions.

The comparison between DFO:trust and DFO:lmlib is not quite clear yet by using problems of 2 and 4 dimensions. On one hand, the results in Tables 4.1 and 4.3, 4.4 and 4.5 show no significant differences in terms of the number of the iterations and the number of the function evaluations. On

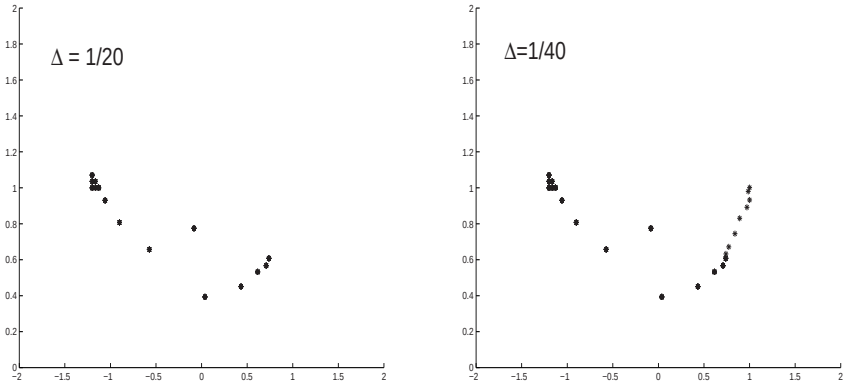

Figure 4.4: Trace of the trust region center on Rosenbrock's function

ε_Δ	#iter	#fun	x^*	$f(x^*)$
1e-2	62	164	(0.0176, -0.0018, 0.0076, 0.0072)	1.3865e-006
1e-3	80	201	(0.0149, -0.0015, 0.0073, 0.0073)	1.0074e-007
1e-4	95	221	(0.0143, -0.0014, 0.0070, 0.0070)	8.5674e-008
1e-5	105	232	(0.0142, -0.0014, 0.0070, 0.0070)	8.4182e-008

Table 4.4: DFO:trust on Powell's function.

the other hand, while solving the trust region subproblem (4.1) on page 83, DFO:trust gives numerical warning messages but DFO:lmlib optimizes without difficulty. The same happened when solving the additional test problems S350, S351, S370 and S371. The results are listed in Table 4.6. The initial point is set to as following.

ε_Δ	#iter	#fun	$x*$	$f(x^*)$
1e-2	69	194	(0.0267, -0.0027, 0.0111, 0.0112)	9.6901e-007
1e-3	88	233	(0.0116, -0.0012, 0.0056, 0.0056)	3.6131e-008
1e-4	100	250	(0.0108, -0.0011, 0.0055, 0.0055)	2.8830e-008
1e-5	109	258	(0.0108, -0.0011, 0.0055, 0.0055)	2.8588e-008

Table 4.5: DFO:lmlib on Powell's function.

S350: $x^0 = (0.25, 0.39, 0.415, 0.39)$,

S351: $x^0 = (2.7, 90, 1500, 10)$,

S370: $x^0 = (0, 0, 0, 0, 0, 0)$,

S371: $x^0 = (0, 0, 0, 0, 0, 0, 0, 0, 0)$.

The initial trust region radius is set to 1. The minimum trust region radius is set to 10^{-3}.

The testing on problem S391 is difficult. We first employed a starting point from the literature that obtained by the following code.

```
for i=1:30
   temp1 = 0;
   for j = 1:30
     temp = (sin(log(sqrt(i/j))))^5+ (cos(log(sqrt(i/j))))^5;
```

		DFO:trust			DFO:lmlib		
problem.	Dim	#iter	#fun	$f(x^*)$	#iter	#fun	$f(x^*)$
s350	4	237	588	0.0016	90	238	0.0077
s351	4	757	2005	319.7963	784	1984	319.7254
s370	6	87	320	0.0657	91	289	0.3394
s371	9	129	638	0.2795	104	532	0.2708

Table 4.6: DFO on problems S350, S351, S370 and S371.

```
      temp1 = temp1 + sqrt(i/j)*temp;
    end
    x(i) = -2.8742711*((i-15)^3+temp1);
end;
```

After two iterates, we were running into numerical problem due to the matrix (2.5) becomes singular to working precision. Then we took the origin as the starting point. The program proceeded successfully towards better function values, but it couldn't give an optimal solution. Indeed, this problem is unbounded, along the line $(-1e+12, 0, \cdots, 0)$ the objective tends to minus infinity.

4.2.4 Conclusions

Our preliminary tests of the DFO algorithm implementation have lead us to the following conclusions:

First, our DFO program proved to be reliable. For different starting points and parameter settings, always the same results were returned by our program.

Second, our DFO program proved to be robust for lower dimension problems. No matter how the trust region radius tolerance changes, there is no significant jump in terms of the number of the iterations and the number of the function evaluations. The largest difficulty we encountered while implementing the DFO algorithm is solving the trust region subproblem. Among the two subroutines, `trust()` and `lmlib()`, described on page 83, `lmlib()` gives better results.

4.3 Build up Scheme

In our approaches for the LJ problem described in Chapter 3, we build up a cluster by adding a randomly generated point, which represents an atom[2], to the optimized smaller cluster configuration.

To understand this build up scheme, we need to know how the LJ function evaluates. We partition the point set into two sets: one set contains the fixed points, another contains the unfixed points. We compute the interaction between the fixed pairs of atoms, the unfixed pairs of atoms and the fixed-unfixed pairs of atoms. So, when we add a randomly generated point to the smaller cluster, the randomly generated point represents an unfixed atom, the points presented in the smaller cluster are treated as fixed point set. Then we minimize the LJ function of the larger system using one of our approaches to optimize the new system.

Note that our working space is in the $n = 3N$ dimensional space, the so-called embedding space. We search for the coordinates of the molecular configuration in the 3D Euclidean space. Let $y^i, \ \ i = 1, \cdots, N, \ y^i \in R^3$ be

[2]In R^3, each point represents the position of an atom. Sometimes we alternately use the two words, depending on the context.

points in the 3D Euclidean space. Then a point in the embedding space is

$$x = VEC(y^1, \cdots, y^N),\ x \in R^{3N}.$$

4.4 Local Search

The local search procedure we used is a MATLAB routine called `fminunc` [MAT] that solves unconstrained problem

$$\min f(x).$$

The `fminunc` implements the BFGS Quasi-Newton method and a trust region method. We set up `options.LargeScale` to be 'off' to switch to the former.

4.5 Visualizing the Structure of a Given Atomic Cluster

We visualize the structure of a given cluster by VRML v2.0 [VRM]. Given a set of points in 3D, we generate automatically a .wrl file by calling procedure `genvrml()`. Then, using VRML, we visualize the structure in 3D.

Chapter 5

Computational Results

In this chapter, we report the results of our various approaches to the LJ atomic cluster problem.

Using Approach I described on page 63, the global minimum potential energy configuration of the small micro-clusters $2 \leq N \leq 5$ are generated. The resulting global minimum values are listed in Table 5.1. The global minimum structure for $N = 2, V^*(2) = -1.000000$ corresponds to 2 atoms on a line. For $N = 3, V^*(3) = -3$, three atoms form a unit equilateral triangle at the global minimum. For $N = 4, V^*(4) = -6$, the four atoms are placed at the vertices of a regular tetrahedron. For $N = 5, V^*(5) = -9.103852$, five atoms are placed at the vertices of a trigonal bi-pyramid, corresponds to

N	2	3	4	5	6
Energy	-1.000000	-3.000000	-6.000000	-9.103852	-12.302927
N	7	8	9	10	
Energy	-15.533062	-19.219111	-23.005312	-25.3367	

Table 5.1: Results of Approach I

the global minimum energy structure. We analyze these structures using the software VRML. For the clusters of $6 \leq N \leq 10$, with Approach I on page 63, our method trapped at local minimums. Test results for $6 \leq N \leq 10$ are listed in Table 5.1.

Tables 5.2 to 5.4 present the results of Approach II described on page 64 and III described on page 66 . The second and third column in Tables 5.2 to 5.4 list the results of Approach II and III, respectively. We compare these results to the ones obtained using gradient based methods: the steepest descend, the BFGS and the DFP Quasi-Newton methods that are listed in the forth, fifth and sixth column in Tables 5.2 to 5.4, respectively. Those results were obtained using the same MATLAB routine as described in Section 4.4. We set the option "HessUpdate" to be "steepdesc" or "bfgs" or "dfp" for the steepest descend, the BFGS and the DFP Quasi-Newton methods, re-

spectively. The maximum number of the function evaluations is set to $2x10^4$. The tolerance of the function improvement is set to 10^{-6}.

We also compare these results with the ones based on LGO [Pin95] branch and bound and random sampling methods that are listed in the seventh and eighth columns of Tables 5.2 to 5.4, respectively. The LGO results were obtained by running LGO on an Intel Pentium III 677MHz RAM 128MB PC. LGO B&B results are obtained using the "automatic branch & bound followed by local search" option. LGO Rd results are obtained using the "automatic random sampling followed by local search" option and running the program 10 times. The data in Tables 5.2 to 5.4 are the best solution of the 10 runs. The global search termination criteria parameter `G_MAXFCT`, that indicates the maximum number of the merit function evaluations allowed in the global scope search, is set to $2x10^4$. The global search termination criteria parameter `MAX_NOSUC`, the maximum number of iterations during which the current best solution has no improvement, is set to 10^4. The local search termination criteria parameter `FI_TOL`, the tolerance of merit function improvement, is set to 10^{-6}.

Finally, in the last columns of Tables 5.2 to 5.4, we give the best known solution reported in the literature [HP71] [FdFRT85] [Wil87] [Nor87].

N	App II	App III.	Steep	BFGS	DFP	LGO BB	LGO Rd	Ref. E
6	-12.3029	-12.7120	-12.3029	-12.3029	-12.3029	-12.7121	-12.3029	-12.7121
7	-15.5331	-16.5054	-15.4994	-15.4995	-15.4995	-15.5330	-15.5331	-16.5054
8	-19.8215	-19.8215	-18.8287	-18.8287	-18.8287	-18.7782	-19.8215	-19.8215
9	-24.1134	-24.1134	-22.1807	-22.1807	-22.1807	-23.0435	-22.1557	-24.1134
10	-27.5452	-28.4225	-26.9425	-26.9548	-26.9385	-27.2739	-26.6226	-28.4225
11	-32.7660	-32.7660	-32.7660	-32.7660	-32.7660	-29.9968	-30.9539	-32.7660
12	-36.1779	-37.9676	-37.9676	-37.9676	-37.9676	-35.1709	-35.4619	-37.9676
13	-41.3944	-44.3268	-44.3268	-44.3268	-44.3268	-37.2377	-41.3559	-44.3268

Table 5.2: Results of Approach II and III

N	App II	App III.	Steep	BFGS	DFP	LGO BB	LGO Rd	Ref. E
14	-47.8452	-47.8452	-47.8451	-47.8451	-47.8451	-44.8740	-46.9347	-47.8452
15	-52.3226	-52.3226	-52.3226	-52.3226	-52.3226	-49.2217	-47.5827	-52.3226
16	-56.8157	-56.8157	-53.9207	-53.9205	-53.9206	-54.7392	-54.4974	-56.8157
17	-61.3180	-61.3180	-58.1264	-58.1252	-58.1258	-52.5670	-58.3596	-61.3180
18	-63.5789	-66.5309	-63.4481	-63.4536	-63.4482	-64.0221	-63.6410	-66.5309
19	-68.6513	-72.6597	-67.5546	-67.5506	-67.5660	-63.5344	-67.7723	-72.6597
20	-77.1770	-77.1770	-73.5189	-73.5177	-73.5243	-71.5136	-73.1039	-77.1770
21	-81.6515	-81.6845	-80.8266	-80.8263	-80.8268	-80.7297	-74.9912	-81.6845
22	-85.2741	-86.8098	-86.0942	-86.0942	-86.0942	-80.2079	-79.8115	-86.8098

Table 5.3: Results of Approach II and III (cont.)

N	App II	App III.	Steep	BFGS	DFP	LGO BB	LGO Rd	Ref. E
23	-87.2745	-92.8445	-90.1185	-90.1202	-90.1255	-83.8708	-84.8686	-92.8445
24	-93.4788	-97.3488	-90.8644	-90.8552	-90.8556	-86.8791	-84.8686	-97.3488
25	-102.3727	-102.3727	-100.0095	-100.0025	-100.0415	-92.1911	-93.0365	-102.3727
26	-105.9982	-108.3156	-106.1089	-106.1089	-106.1089	-97.4813	-100.4841	-108.3156
27	-112.8255	-112.8735	-110.1762	-110.1754	-110.1861	-104.5642	-101.2285	-112.8735
28	-117.7780	-117.8224	-97.0384	-96.9612	-97.0102	-105.9055	-102.6110	-117.8224
29	-121.0798	-123.5873	-121.8190	-121.8166	-121.8158	-109.9146	-110.0736	-123.5873
30	-126.9484	-128.2865	-116.9918	-116.9856	-117.0054	-116.0006	-117.5743	-128.2865

Table 5.4: Results of Approach II and III (cont.)

It is easy to see that all the three gradient based methods can only locate the best known solution of the clusters of $N = 11, 12$ and 13. LGO branch and bound method can catch the cluster of $N = 6$ and the random sampling method can catch the cluster of $N = 8$. Our approach III located the best known solution of all the 29 cases.

Approach III may be extensible to solve larger cluster. Due to the limit of the time, we were not allowed to do exhaustive test on the larger clusters.

The results of Approach IV described on page 67, the DFO based approach, are listed in Table 5.5. We tested for the cluster of N up to 13 on IBM RS6000 workstation. As we already know from Section 2.3 that in the DFO algorithm, the matrix (2.5) may become ill conditioned, while updating the interpolation point set, thus the linear system (2.4) becomes very difficult to solve. This has happened when the dimension of the problem becomes larger. Indeed, for $N = 7$ (dimension of 21), we have received the warning message that "matrix is singular to working precision" while solving the linear system (2.4). However, due to the self-recovering property of the DFO algorithm, the error may be corrected from iterate to iterate.

Comparing the results of Approach IV in Table 5.5 with the best known solution reported in the literature listed in Tables 5.2 to 5.4, it can be seen

N	2	3	4	5	6	7
Energy	-1.0000	-3.0000	-6.0000	-9.1039	-12.7121	-16.5054
N	8	9	10	11	12	13
Energy	-19.8215	-24.1134	-27.5559	-31.9147	-36.2430	-44.3268

Table 5.5: Results of Approach IV

that the global minima of the clusters of $N = 2, \ldots, 10$ and 13 have been located. For larger N, the size of the problem limits the use of Approach IV. This limitation may come from the DFO algorithm. Indeed, for a full quadratic interpolation, DFO needs $\frac{(n+1)(n+2)}{2}$ points. For instance, if we consider a cluster of 15 atoms, the dimension of the embedding space (see page 100) would be 45. The number of the points with which we deal is 1081, and that is the size of the linear system (2.4) to be solved at each iteration. This is a completely dense system. Thus it needs a lot of memory to store and manipulate it. Further, its numerical properties deteriorate as the algorithm progresses. Moreover, the geometry of the quadratic model for higher dimensional problems, especially for non smooth functions such as the LJ potential energy function is very hard to keep poised while updating the interpolation point set. A non-poised interpolation point set results numerical problems

therefore terminate the DFO algorithm.

Chapter 6

Conclusions and Perspectives

In this thesis, a deterministic global optimization approach, that is a combination of direct search methods with local search algorithms, was introduced to determine the global minima of the potential energy surfaces of LJ atomic clusters. Our computational results are presented up to 30 atoms. All the best known optimal results from the literature have been reproduced.

In many applications, searching for optimal solutions is aided by using "heuristic" information. In the case of the LJ atomic cluster problem, we adapt growth (build up) rules [FdFRT85], when searching for the optimal structure of a larger cluster by adding new atoms to an already optimized smaller structures.

Two direct search methods have been used in our approaches. The first is a geometry-based direct search method, the Nelder-Mead simplex method, that uses a simple function decrease principle. At each iteration, the simplex is modified so that the simplex can change shape and thereby "adapt itself to the local landscape" [NM65]. With Approach III on page 66, when the Nelder-Mead simplex method is enhanced with a build-up initialization procedure and combined with a BFGS local search procedure, the global optima of the LJ atomic clusters up to 30 atoms have been located.

The second is a model-based direct search method, called DFO, that uses the function values to build a convenient quadratic model through interpolation. The main difficulty in practice with this method is that in higher dimensions, the geometry of the underlying model may deteriorate while updating the interpolation point set, especially if the original function is not "nice", not smooth. With Approach IV on page 67, we rediscovered the global optima of the LJ atomic clusters up to 13 atoms.

The proposed approaches were implemented in MATLAB. We provided a full implementation of the Nelder-Mead simplex method, as well as the DFO algorithm. Since our approaches are implemented in MATLAB, memory and speed of computations limit the size of solvable problems.

From the results presented in Chapter 5, it is possible to infer that the combination of direct search methods with local search algorithms in LJ atomic cluster optimization is successful in locating the global optimum, avoiding being trapped in local optimum. This approach may lead to very promising algorithms for molecule geometry optimization. It would be interesting to apply it to other potential minimization problems as well.

As a concluding remark we note that this research intended to propose a general method capable of discovering all optima in the molecule geometry optimization. Future computational effort should concentrate on improving the efficiency of the algorithm and exploring the possibility of parallelizing energy computations that can be used to solve larger problems.

Bibliography

[All85] N.L. Allinger. Molecular mechanics. In *International School of Crystallography:Static and Dynamic Implications of Precise Structural Information*, pages 149–164. Italy, 1985.

[BH86] R. Biswas and D.R. Hamann. Simulated annealing of silicon atom clusters in langevin molecular dynamics. *Phys. Rev. B*, (34):895–901, 1986.

[BSS93] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming. Theory and Algorithms.* John Wiley & Sons Ltd, New York, 1993.

[CST97a] A.R. Conn, K. Scheinberg, and Ph.L. Toint. On the convergence of derivative-free methods for unconstrained optimization. In A. Iserles and M. Buhmann, editors, *Approximation Theory and*

Optimization: Tributes to M.J.D. Powell, pages 83–108. Cambridge University Press, Cambridge, UK, 1997.

[CST97b] A.R. Conn, K. Scheinberg, and Ph.L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Math. Programming*, (79):397–414, 1997.

[CSW94] T.F. Coleman, D. Shalloway, and Z. Wu. A parallel build-up algorithm for global energy minimizations of molecular clusters using effective energy simulated annealing. *J. Global Optim.*, (4):171–185, 1994.

[Dav91] W.C. Davidon. Variable metric method for minimization. *SIAM J. Optimization*, (1):1–17, 1991.

[DT91] J.E. Dennis and V. Torczon. Direct search methods on parallel machines. *SIAM J. Optim.*, (1):448–474, 1991.

[DTMH96a] D.M. Deaven, N. Tit, J.R. Morris, and K.M. Ho. Structural optimization of Lennard-Jones clusters by a genetic algorithm. *Chemical Physics Letters*, (256):195–, 1996.

[DTMH96b] D.N. Deaven, N. Tit, J.R. Morris, and K.M. Ho. Structural optimization of Lennard-Jones clusters by a genetic algorithm. *Chem. Phy. Lett.*, (256):195, 1996.

[DW96] J.P. Doucet and J. Weber. *Computer-Aided Molecular Design: Theory and Applications*. Harcour Brace & Company, London, 1996.

[FdFRT85] J. Farges, M.F. de Feraudy, B. Raoult, and G. Torchet. Cluster models made of double icosahedron units. *Surf. Sci.*, (156):370–, 1985.

[Fin96] L. Finschi. An implementation of the Levenberg-Marquardt algorithm. Institut fuer Operations Research, Clausiusstrasse 45, CH-8092 Zuerich, 1996.

[Fle87] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons Ltd, 2^{nd} edition, 1987.

[FZ99] B. Fan and R. Zhang. *Introduction to Computer Chemistry*. Lanzhou University Press, Lanzhou, China, 1999.

[Gay81] D.M. Gay. Computing optimal locally constrained steps. *SIAM J. Sci. Stat. Comput.*, 2(2):186–197, 6 1981.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[GKS97] M.S. Gockenbach, A.J. Kearsley, and W.W. Symes. An infeasible point method for minimizing the Lennard-Jones potential. *Computational Optimization and Applications*, (8):273–286, 1997.

[GL89] G.H. Golub and C.F. Van Loan. *Matrix Computation.* The Johns Hopkins University Press, London, 2^{nd} edition, 1989.

[HJ61] R. Hooke and T.A. Jeeves. "Direct search" solution of numerical and statistical problems. *Journal of the ACM*, (8):212–229, 1961.

[HP63] K.B. Harvey and G.B. Porter. *Introduction to Physical Inorganic Chemistry.* Addison-Wesley Publishing Company, Inc., London, 1963.

[HP71] M.R. Hoare and P. Pal. Physical cluster mechanics: Statics and energy surfaces for monatomic systems. *Adv. Phys.*, (20):161–195, 1971.

[KGV83] S. Kirkpatrick, C.D. Gelatt, and M. Vecchi. *Science*, (220):671–680, 1983.

[LRWW98] J.C. Lagarias, J.A. Reeds, M.H. Wright, and P.E. Wright. Convergence properties of the Nelder-Mead simplex algorithm in low dimensions. *SIAM J. Optimization*, (9):112–147, 1998.

[LS02] M. Locatelli and F. Schoen. Fast global optimization of difficult Lennard-Jones clusters. *Computational Optimization and Applications*, (21):55–70, 2002.

[MAT] MATLAB v6.0 toolbox. MATHWorks Inc.

[MM94] J.C. Meza and M.L. Martinez. On the use of direct search methods for the molecular conformation problem. *Journal of Computational Chemistry*, (15):627–632, 1994.

[MRR+58] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *J. Chem. Phys.*, (21):1087–1092, 1958.

[MS83] J.J. Moré and D.C. Sorenson. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, (4):553–572, 1983.

[MW95] J.J. Moré and Z. Wu. Global continuation for distance geometry ploblems. Preprint, 1995.

[NM65] J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, (7):308–313, 1965.

[Nor87] J.A. Northby. Structure and binding of Lennard-Jones clusters: $13 \leq n \leq 147$. *J. Chem. Phys.*, (87):6166, 1987.

[NW99] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, New York, 1999.

[Pin70] M. Pincus. A monte carlo method for the approximate solution of certain types of constrained optimization problems. *Oper. Res.*, (18):1225–1228, 1970.

[Pin95] J.D. Pintér. LGO: An implementation of a Lipschitz global optimization procedure. Research Report NM-R9522, National Resaerch Institute for Mathematics and Computer Science (CWI), Amsterdam, 1995.

[Pin96] J.D. Pintér. *Global Optimization in Action (Continuous and Lipschitz Optimization: Algorithms, Implementation and Applications*. Kluwer Academic Publishers, Boston, 1996.

[PKS89] L. Piela, J. Kostrowick, and H.A. Sheraga. The multiple-minima problem in the conformational analysis of molecules: Deformation of the protein energy supersurface by the diffusion equation method. *J. Phy. Chem.*, (93):3339–3346, 1989.

[Pow94a] M.J.D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis, Proceeding of the 6th Workshop on Optimization and Numerical Analysis*, volume 275, pages 5–67. Kluwer Academic Publishers, Oaxaca, Mexico, 1994.

[Pow94b] M.J.D. Powell. A direct search optimization method that models the objective by quadraic interpolation. Presentation at the 5th Stockholm Optimization Days, 1994.

[Pow96] M.J.D. Powell. Trust region methods that employ quadratic interpolation to the objective function. presentation at the 5th SIAM Confrence on Optimization, 1996.

[Sch00] K. Scheinberg. Derivative free optimization method. Preprint, 2000.

[Sha92] D. Shalloway. Packet annealing: A deterministic method for global minimization, application to molecular conformation. In *Recent Advances in Global Optimization*, pages 433–477. Princeton University Press, 1992.

[SHH62] W. Spendley, G.R. Hext, and F.R. Himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, (4):441–461, 1962.

[SOP] Standard optimization problems in AMPL software package. http://www.ampl.com/.

[Ste98] G.W. Stewart. *Matrix Algorithms*, volume II. SIAM, 1998.

[Str94] J.E. Straub. *Optimization Techniques with Applications to Proteins*. Boston University, Boston, Massachusetts, 1994.

[SX95] Th. Sauer and Y. Xu. On multivariate Lagrange interpolation. *Mathematics of Computation*, (64):1147–1170, 1995.

[Ter00] T. Terlaky. Continuous optimization algorithms. Lecture notes. http://www.cas.mcmaster.ca/∼terlaky, 2000.

[Tor89] V.J. Torczon. *Multi-directional Search: A Direct Search Algorithm for Parallel Machines*. Ph.D. Thesis, Rice University, Houston, Texas, 1989.

[VRM] Virtual Reality Modelling Language, a 3d web browser. http://www.vrml.org/.

[WC90] S.R. Wilson and W. Cui. Applications of simulated annealing to peptides. *Biopolymers*, (29):225–235, 1990.

[WCMS88] S.R. Wilson, W. Cui, J.W. Moskowitz, and K.E. Schmidt. Conformational analysis of flexible molecules: Location of the

global minimum energy conformation by the simulated annealing method. *Tetrahedron Letters*, 29(35):4373–4376, 1988.

[WD97] 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. *Chem. Phys. Lett.*, (269):408–412, 1997.

[Wil87] L.T. Wille. Minimum-energy configurations of atomic clusters: New results obtained by simulated annealing. *Chem. Phy. Lett.*, (133):405–410, 1987.

[Win69] D. Winfield. *Function and Functional Optimization by Interpolation in Data Tables*. Ph.D. Thesis, Harvard University, Cambridge, MA, 1969.

[Win73] D. Winfield. Function minimization by interpolation in data tables. *Journal of the Institute of Mathematics and its Applications*, (12):339–347, 1973.

[Wri96] M.H. Wright. Direct search methods: once scorned, now respectable. In *Numerical Analysis 1995*, pages 191–208. Addison Wesley Longman, Harlow, United Kingdom, 1996.

[Wri00] M.H. Wright. What, if anything, is new in optimization? In J.M. Ball and J.C.R. Hunt, editors, *ICIAM 99 - Proceedings of the Fourth International Congress on Industrial & Applied Mathematics, Edinburgh*, pages 259–270. Oxford University Press, United Kingdom, 2000.

[WV85] L.T. Wille and J. Vennik. Computational complexity of the ground-state determination of atomic clusters. *J. Phys.*, A(18):L419, 1985.