MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Genetic algorithms to optimize polarizable force fields in order to reproduce infrared spectra"

verfasst von / submitted by

Aleksandar Doknic, BSc MSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna, 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 910
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Computational Science
Betreut von / Supervisor:	Assoz. Prof. Dipl.-Chem. Dr. Christian Schröder

Acknowledgements

I am profoundly grateful to Christian Schröder and Florian Jörg for their excellent collaboration and support, and for making this project and thesis possible.

I also would like to thank the University of Vienna for offering the Computational Science degree program and promoting interdisciplinary work, and the State of Austria for making university education accessible and affordable.

Finally, I would like to thank my parents, Milutin and Slobodanka, for supporting my scientific ambitions.

Kurzfassung

Protische ionische Flüssigkeiten tauschen Protonen zwischen den Kationen und den Anionen aus, dabei sind jedoch die experimentellen Daten vom Equilibrium zwischen geladenen und ungeladenen Molekülen nicht eindeutig, was weitere Untersuchungen erforderlich macht. Solche Untersuchungen können mithilfe der Infrarot Spektroskopie durchgeführt werden. Derzeitige polarisierbare Kraftfelder erbringen jedoch eine schlechte Leistung bei der Vorhersage von Infrarotspektren. Basierend auf der Parametrisierungssoftware AFMM und inspiriert vom hybriden, genetischen Optimierungsansatz von Paramfit, wird ein neuartiger hybrider genetischer Algorithmus für die Optimierung polarisierbarer Kraftfelder in molekulardynamischen Simulationen entwickelt und implementiert. Nach dem Definieren der Zielfunktion, basierend auf dem Vergleich von Normalmoden, und der Analyse der Optimierungsoberflächen von sechs verschiedenen Molekülen durch die Methode des achsenparallelen Schneidens, wird der Algorithmus für die Problemstellung angepasst. Laufzeitexperimente testen die Leistungsfähigkeit des Optimierungsalgorithmus mit verschiedenen Parametereinstellungen (z.B. Populationsgröße) an sechs verschiedenen Molekülen und zwei Kraftfeldarten. Es zeigt sich, dass die optimierten Kraftfelder eine bessere Leistung bei der Annäherung von quantenmechanisch berechneten Normalmoden erbringen als frühere Kraftfelder. Es zeigt sich allerdings auch, dass für eine genauere Reproduktion der Infrarotspektren aus den experimentellen Daten weitere Untersuchungen erforderlich sind.

Abstract

Protic ionic liquids exchange protons between cations and anions, but experimental data on the equilibrium between charged and uncharged molecules is ambiguous and deserves further investigation. Such investigations can be performed with the help of infrared spectroscopy. However, current polarizable force fields perform poorly at predicting infrared spectra. Based on the parameter fitting tool AFMM, and inspired by the hybrid genetic optimization approach of Paramfit, a novel hybrid genetic algorithm for the purpose of optimizing polarizable force fields for molecular dynamics simulations is developed and implemented. After defining a fitness function based on matching normal modes, and analyzing the optimization surface for six different molecules by using the approach of random axis-parallel slicing, the algorithm is adapted for the given problem. Runtime experiments assess the performance of the optimization algorithm with different parameter settings (e.g. population size) on six different molecules and two types of force fields (CHARMM and CLaP). It is shown that optimized force fields perform better at computing normal modes aligning with quantum mechanically calculated normal modes than previous force fields, but that reproducing infrared spectra that align more accurately with the experimental data will require further research.

Contents

CHAPTER 1

Introduction

Over the last decades computer simulations have become an essential tool in chemistry, biology, and many other fields. Their importance as instruments of science is comparable to the importance of telescopes in astronomy or microscopes in biology. However, traditional microscopes have limits. One of the limits is the fact that most molecules are too small to be seen with optical instruments, because single molecules exist in a different scale of space (distances are commonly measured in Ångström: 10^{-10} m) and time (significant relative movements in less than 10^{-9} s) than humans. Most observable measures like temperature and entropy are therefore time-averaged measures for large samples of molecules.

There are advanced techniques that can be used to indirectly analyze the structure and geometry of molecules and even atoms. Such techniques are often time-consuming, expensive, or suffer from noise caused by technical limitations or by physical limitations like the Pauli principle. For example, **nuclear magnetic resonance** (NMR) spectroscopy observes the nuclear response after applying an external magnetic field, and **electron paramagnetic resonance** (EPR or electron spin resonance, ESR) spectroscopy observes the electron response [1, Ch.12]. **X-ray diffraction** in crystallography is used to determine the arrangement of atoms in periodic crystals. **Neutron diffraction** interacts with the nucleus and **electron diffraction** is commonly used for analyzing surfaces of solids [1, Ch.15]. **Infrared spectroscopy** or **vibrational spectroscopy** is one of the most common ways to analyze organic compounds due to its capability to detect and distinguish functional groups.

Observing processes on a molecular level is a challenge. Computer simulations can avoid many of these problems. *Ab initio* simulations based on quantum mechanics can, in theory, accurately reproduce real-world behavior of atoms and molecules. However, solving the **Schroedinger's equation** gets exponentially more time consuming for larger systems and computations over larger time scales become difficult [2, p. 165]. **Molecular mechanics** (MM) [2] is based on Newtonian mechanics and allows computationally cheap simulations based on solving **Newton's equation of motion** for particles represented by point masses. Positions and velocities of the particles are computed from the **total interaction potential** that contains approximations of all relevant intra- and intermolecular interactions and requires fitting **force constants** for each system of particles. Monte Carlo-like algorithms are commonly used to parametrize force fields, but they turned out to be very time consuming. The goal of this thesis is to explore the suitability of **genetic algorithms** for parametrizing force fields and verifying the results for a selected number of ionic liquid [3] compounds by reproducing the respective (QM-calculated) infrared spectra.

This thesis is written as a part of my **Computational Science** studies at the Faculty of Physics at the University of Vienna under the supervision of Christian Schröder from the Department of Computational Biological Chemistry. The field of Computational Science lies at the intersection between mathematics, computer science, and domain sciences and deals with the challenges that come with the application of computers for scientific purposes.

1.1 The project

Ionic liquids (IL) are liquids that only exist in ionic form, typically as salts with a melting point below 100°C [3][4]. Some IL even have their melting point at room temperature and are called room temperature ionic liquids (RTIL). There is a large variety of possible IL with interesting properties that could be useful for different applications, especially as green solvents. However, producing them is relatively expensive and contamination is difficult to prevent. The research team around Christian Schröder is using computer simulations to study the behavior of IL and to screen for promising candidates.

For this purpose molecular dynamics simulation tools, such as **CHARMM** [5], can be used. However, in order to run a simulation and compute the molecular forces it is necessary to set a number of parameters that represent the strength of the molecular interactions, the so called **force constants**. These parameters vary for every molecule and molecular system. Molecular dynamics simulations can run on larger (biologically relevant) time scales, larger spatial scales, and a larger number of particles than the more computationally expensive quantum mechanical simulations. However, this is only possible with the proper force constants.

In order to confirm that the parameters actually lead to a simulation that approximates real world behavior of the compound, it is possible to compare the measured **infrared absorption spectrum** (IR spectrum) of a synthesized IL with the MD simulated absorption spectrum. The assumption is that a good matching of both spectra indicates a good approximation. In our case the target spectra will be generated from quantum mechanical simulations based on solving Schroedinger's equations.

The goal of this project is to compute better intramolecular force constants that lead to absorption spectra similar to the QM-calculated spectra. Since computing the IR spectra takes a long time, the vibrational **normal modes** are used as a measure of fitness for a set of parameters during the optimization process.

1.2 Contributions

The optimization landscapes for different anions and cations have been sampled, visualized, and analyzed. A hybrid genetic algorithm in combination with a local optimization algorithm was designed and implemented. The success of this approach was shown by finding new parameters with significantly improved fitness compared to the reference parameters, and by showing a better overlap of the MM-calculated spectra with the QM-calculated spectra.

CHAPTER 2

Background

This chapter provides a concise introduction into key aspects of molecular dynamics, force fields, ionic liquids, and other topics which are relevant for the following chapters. Genetic algorithms will be discussed in the next chapter.

2.1 Molecular mechanics

Molecular mechanics is based on the Born-Oppenheimer approximation which states that electrons quickly adapt to the movement of nuclei due to the fact that the proton mass is approximately 1836 times [6] the electron mass. Therefore it is possible to achieve a reasonable accuracy by only considering the nuclei coordinates in calculations [2, Ch.2, Ch.4]. Additionally, quantum mechanical effects are less noticeable for objects with larger momentum (since the wavelength converges to 0 according to the de Broglie relation $\lambda = h/p$).

2.2 Force fields

Molecular dynamics simulations require the parametrized interaction potential function $U(\vec{R})$ to derive the particle forces, velocities, and eventually positions.

$$U(\vec{R}) = \text{intramolecular potentials} + \text{intermolecular potentials} \tag{2.1}$$

The total interaction potential consists of intramolecular potentials and intermolecular potentials. Intramolecular potentials describe the interactions between atoms which are connected by at most three bonded neighbors. In most cases they consist of stretching potentials between pairs of two atoms, angle bending potentials in systems of three atoms, and torsional terms in systems of four atoms, see figure 2.1. Sometimes the force fields

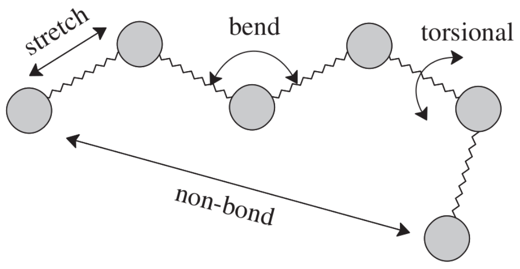


Figure 2.1: The fundamental force field energy terms: bond stretching, angle bending, and torsion. Figure from [7].

include improper (out of plane) terms in a system of four atoms. On the other hand there are intermolecular interactions: the electrostatic interactions are represented by the Coulomb potential and the Van der Waals interactions are described by the Lenard Jones potential. Sometimes external potentials are also added, but this is not considered here.

The original CHARMM publication describes the following interaction potential [5]:

$$U(\vec{R}) = \sum_{bonds} K_b(b-b_0)^2 + \sum_{angles} K_\theta(\theta-\theta_0)^2 + \sum_{Urey-Bradley} K_{UB}(S-S_0)^2 + \sum_{dihedrals} K_\phi(1+cos(n\phi-\delta))^2 + \sum_{impropers} K_\omega(\omega-\omega_0)^2 + \sum_{non-bonded}\left\{\epsilon_{ij}^{min}\left[\left(\frac{R_{ij}^{min}}{r_{ij}}\right)^{12} - 2\left(\frac{R_{ij}^{min}}{r_{ij}}\right)^{6}\right] + \frac{q_i q_j}{4\pi\epsilon_0\epsilon r_{ij}}\right\} \tag{2.2}$$

The Urey-Bradley potential was listed for completeness, but is not used here. The intramolecular potentials in equation 2.2, with exception of the dihedral interactions, resemble a harmonic oscillator. The potential energy increases with the squared distance from the equilibrium position, which is denoted by a small 0. The K_* value is the force constant and needs to be chosen well in order to produce accurate results.

The potential energy $V(x)$ with $x = R - R_e$ (displacement from equilibrium R_e, see figure 2.2) can be expanded around its minimum. [1, Ch.11]:

$$V(x) = V(0) + \left(\frac{dV}{dx}\right)_0 x + \frac{1}{2}\left(\frac{d^2V}{dx^2}\right)_0 x^2 + ... \tag{2.3}$$

V(0) can be set to 0 and the first derivative is 0 at the minimum. The first surviving term offers a simple approximation:

$$\begin{aligned} V(x) &= \frac{1}{2}\left(\frac{d^2V}{dx^2}\right)_0 x^2 \\ V(x) &= \frac{1}{2}Kx^2 \\ V(x) &= \frac{1}{2}K(R-R_e)^2 \end{aligned} \tag{2.4}$$

K is defined as the **force constant** and it describes the strength of a bond.

2.2.1 Bond stretching

The **Morse potential** is a popular model for describing energies in varying distances between bonded atoms. It includes the repulsion at close distances and the dissociation energies at far distances. The minimum R_e represents the low energy or equilibrium state that occurs when all other terms are set to zero [2, Ch.4]:

$$u_{morse}(l) = D_e(1 - e^{-a(b-b_0)})^2 \tag{2.5}$$

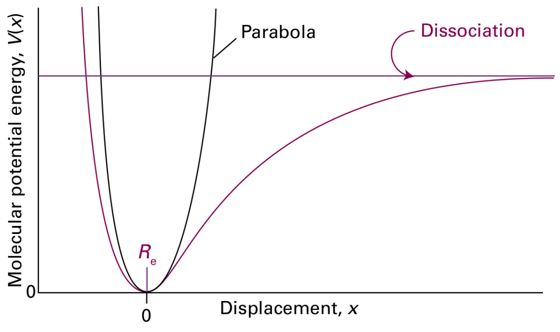


Figure 2.2: Shapes of Morse potential (red) and parabola-shaped harmonic potential. Image from [1, Ch.11].

Figure 2.2 shows the shape of the Morse potential and a parabolic approximation based on Hooke's law. For a more accurate force field it is possible to add more terms from a Taylor expansion of the Morse potential. The harmonic potential is a simple second order approximation, but it does not take into consideration the dissociation energy which occurs at larger displacements (= high energy levels). However, close to the equilibrium the approximation works reasonably well [2, Ch.4].

$$u(b) = \frac{1}{2}K_b(b-b_0)^2 \tag{2.6}$$

One advantage of using the harmonic model is that the derivative (the force) is simply $F(b) = K_b(b - b_0)$. K_b is the force constant, or bond stretching coefficient. It describes the magnitude of the bond stretching energy. The force constants are measured in $\frac{\text{kcal}}{\text{mol}\text{Å}^2}$ [2, Ch.4].

2.2.2 Angle bending

Angle bending is frequently represented by a harmonic potential similar to the parabolic function in figure 2.2 [2, Ch.4]:

$$u(\theta) = K_\theta(\theta - \theta_0)^2 \tag{2.7}$$

The force constant K_θ is measured in $\frac{\text{kcal}}{\text{mol deg}^2}$ and is usually smaller than the bond stretching constants. In other words, angle bending is much more flexible compared to bond stretching. Consequently, angular vibrations have lower frequencies [2, Ch.4].

2.2.3 Torsional terms

Cosine series expansions are used to represent the torsional potentials [2, Ch.4]:

$$u(\phi) = K_\phi(1 + cos(n\phi - \delta))^2 \tag{2.8}$$

The torsion energy changes with the bond rotating angle ϕ. n is the multiplicity, which denotes the number of the maxima during a complete rotation around the central bond, and δ is the phase shift, which moves the cosine potential to the left or right. The force constants of torsional potentials are usually lower compared to angles and bonds. Furthermore, it is not uncommon that during a simulation all dihedral angles are visited as this potential is periodic and has usually low barriers. As a consequence, these potentials result in low frequencies. [5]

2.2.4 Improper torsions

Improper torsion or out-of-plane bending potential can be defined in different ways. Our CHARMM force field in eq. 2.2 uses a harmonic potential:

$$u(\omega) = K_\omega(\omega - \omega_0)^2 \tag{2.9}$$

Improper torsions are usually applied to retain planar geometry of aromatic systems. Hence, force constants can be quite strong and vibrational frequencies are sensitive to the presence of out-of-plane terms [2, Ch.4].

2.2.5 Electrostatic interactions

The electrostatic interaction between atoms i and j is computed by using Coulomb's law [2, Ch.4]:

$$u(r) = \frac{q_i q_j}{4\pi\epsilon_0 \, \epsilon \, r_{ij}} \tag{2.10}$$

ϵ is the static dielectric constant of the solvent and ϵ_0 is the permittivity of vacuum. Usually, ϵ is set to unity in MD simulations with explicit solvents as the manifold of Coulomb interactions between the atoms emulate the overall dielectric constant ϵ. [5]

2.2.6 Van der Waals interactions

The **van der Waals interactions** represent the attractive and repulsive forces between atoms and molecules. Since a large number of van der Waals potentials need to be evaluated, it is desirable to have a simple form. One of the most common forms is the **Lennard-Jones 12-6** function [2, Ch.4]:

$$u(r) = \epsilon_{ij}^{min} \left[\left(\frac{R_{ij}^{min}}{r_{ij}} \right)^{12} - 2 \left(\frac{R_{ij}^{min}}{r_{ij}} \right)^{6} \right] \tag{2.11}$$

The left power term is repulsive while the right power term is attractive. $R_{ij}^{min} = 2^{\frac{1}{6}}\sigma$ is the distance where the Lennard-Jones potential becomes minimal and σ resembles the particle radius. ϵ_{ij}^{min} is the well depth [2, Ch.4].

2.2.7 Induced polarization

Induced polarization can be simulated by using the Drude oscillator model. Virtual drude particles with mass and charge are attached to their parent atoms via harmonic springs, making them polarizable. The mass and charge are subtracted from the parent atom. [8]

2.3 Normal modes

The vibrational motions in molecules can be described by **Hooke's law**: The restoring force acting on a mass increases as the displacement from the equilibrium position x increases [1, Ch.7]:

$$F_x = -k_i x \tag{2.12}$$

Solving newton's equation of motion at x = 0 and t = 0 shows that the particles oscillate harmonically with the frequency ν:

$$m\frac{d^2x}{dt^2} = -k_i x \tag{2.13}$$

$$x(t) = A\sin(2\pi\nu t) \tag{2.14}$$

The independent movements of atoms are called the *normal modes*. Assuming a harmonic model, the normal modes can be computed from the Hessian matrix H of the second derivative of the total potential energy (e.g. equation 2.2) or wave function. Since vibrations depend on mass, the matrix has first to be transformed to the mass-weighted coordinates matrix $F = M^{-1/2}HM^{-1/2}$. M is a 3N × 3N matrix with the particle masses on the diagonal. The eigenvectors and eigenvalues are computed by solving the secular equation $|F - \lambda I| = 0$ and the respective linear systems. Six eigenvalues will be 0, representing the translational and rotational motion. The vibrational frequencies are then calculated from the eigenvalues: $v_i = \frac{\sqrt{\lambda_i}}{2\pi}$. These vibrational frequencies are what we want to compare in the fitness function. The harmonic approximation is acceptable if the amplitude (displacement from equilibrium) of the vibration is not too large. In other words, at very high energy levels (temperature) the approximation fails. [2, Ch.5.8]

2.4 IR spectroscopy

Oscillating electric dipoles generate electromagnetic waves. On the other hand, electric dipoles can be induced by electromagnetic waves. Vibrational transitions in particular correspond to the infrared spectrum. IR spectroscopy makes use of the interaction between infrared radiation and molecular vibrations to detect functional groups. Other wavelengths correspond to other types of state transition (e.g. rotational spectroscopy). [1, Ch.11]

The horizontal axis represents the wavenumber (in cm^{-1}) while the vertical axis represents the absorption. The common IR frequency range is 4000 cm^{-1} to 400 cm^{-1} This corresponds to the wavelength λ from 2.5 to 15 µm. [9]

$$\text{Wavenumber: } \tilde{\text{v}} = \frac{10^4}{\lambda} \tag{2.15}$$

The reproduction of accurate IR spectra with molecular mechanics force fields is a difficult task [2, Ch.4] and contradicts to some extent the common philosophy of force field parametrization. In order to increase the transferability, atoms are gathered by atom type sharing very similar physicochemical behavior. This way, the number of harmonic, dihedral and improper potentials are greatly reduced. However, this strategy prevents extremely high accuracy of MD based normal mode calculations. Additionally, common force fields treat bonds, angles and torsions separately. There is usually no cross-term correlating bond with the respective angle between them.

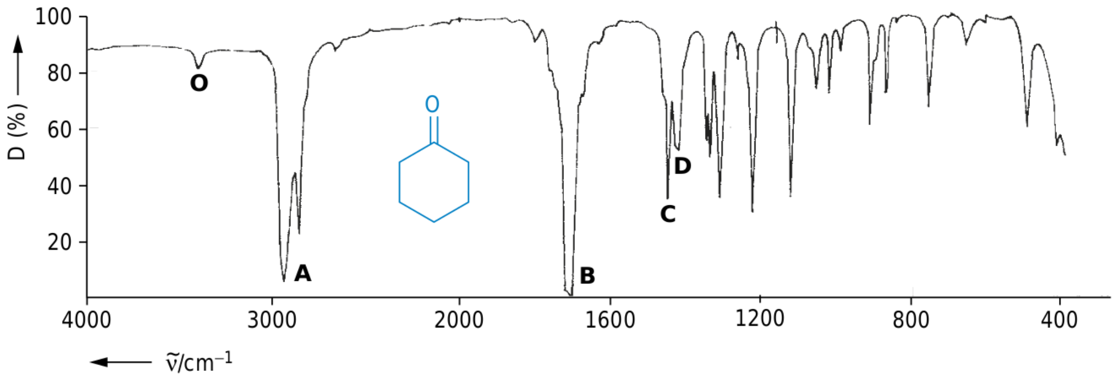


Figure 2.3: IR absorption spectrum of Cyclohexanone. **O** vibrations are caused by the Carbonyl group, **A** vibrations are caused by C-H valence vibrations, and **B** vibrations are created by the double bond C=O. **C** and **D** are deformation vibrations. The right ride of the spectrum is called the **fingerprint region** and is difficult to interpret. Figure from [9].

2.5 Ionic liquids

Ionic liquids (IL) consist of negatively and positively charged ions which act like a liquid, instead of forming a grid-like structure like most salts at low temperatures (below 100°C). IL have useful properties as chemical solvents, e.g. low flammability, low viscosity, and tunable acidity or basicity [4]. The dielectric constant of ionic liquids is lower than in water while the molecules are closer together, which results results in a low vapor pressure [3, Ch.1].

There are different types of IL. **Protic ionic liquids** in particular can be easily created by mixing cations and anions together. They are based on proton transfers between the Brønsted base and acid that oscillates at an equilibrium [10]. One example is
$\mathrm{Im1H^{+} + OAc^{-} = Im1 + HOAc}$.

As IL are charged species, their Coulomb interaction is long-ranged, making large simulation systems inevitable. This becomes out of reach of quantum chemical calculations, in particular if dynamical properties like the conductivity are of interest. The goal of this work is develop molecular dynamics force fields for the cations and anions to reproduce the IR spectra of ionic liquids.

CHAPTER 3

Related work

This chapter describes popular force field optimization tools that provide the foundation for the approach in this thesis. In particular, parts of the AFMM [11] script (with modifications) are used for computing the fitness function. The general hybrid algorithm idea of Paramfit [12] was used as inspiration.

3.1 Automated Frequency Matching Method

AFMM [11] is a vibrational parametrization program for molecular mechanics (MM) simulations in CHARMM. The goal of AFMM is to find parameters for the potential energy function (equation 2.2) for different molecules and systems of molecules. The algorithm is based on a simple Monte Carlo scheme and provides a good starting point for the development of more sophisticated approaches.

For each generated parameter set the quality needs to be tested by a merit function (or fitness function). In AFMM the merit function (equation 3.1) is based on comparing the results of a normal mode analysis (NMA) in CHARMM with some reference normal modes from a quantum mechanical calculation (e.g. from Gaussian 98). The *vibran* module in CHARMM computes the normal modes and its respective eigenvectors and vibrational frequencies for a given set of parameters. These eigenvectors are then projected onto the reference eigenvectors in order to find the best matching between eigenvector pairs. The squared differences between respective vibrational frequencies ν_i and ν_j^{max} are then summed up to generate a merit function. Ω_i^2 is a weight parameter that can be used to create a bias for biologically more relevant frequencies or improved eigenvector projections [11].

In AFMM the merit function is defined as:

$$Y^2 = \sum_{3N-6} \Omega_i^2 (\nu_i - \nu_j^{max})^2 \tag{3.1}$$

For N atoms there exist 3N-6 independent vibrational frequencies (3N-5 for linear molecules). The six degrees of freedom coming from rotation and translation of the system in x, y and z directions are subtracted from the total degrees of freedom (3N).

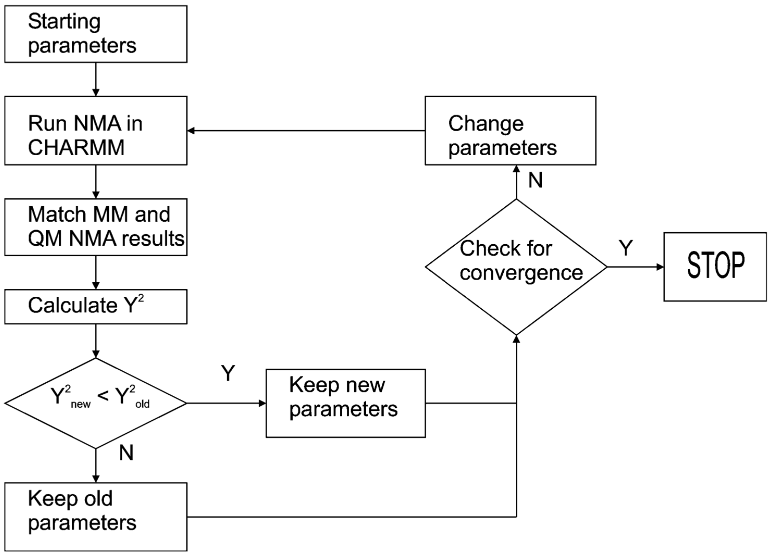


Figure 3.1: AFMM workflow. Image taken from original publication: [11].

Figure 3.1 gives an overview over the parameter fitting procedure: Starting parameters are required to start the procedure. Normal modes are computed and matched with the QM NMA results. The merit function is then evaluated. If a parameter sets with a better merit value is found, the old parameters are overwritten. The loss of previous results is one of the disadvantages of this approach. This is especially problematic since each merit function evaluation requires running a NMA in CHARMM, which is computationally more expensive than only computing a mathematical function. While AFMM works for some systems, it is insufficient for our purpose and converges too slowly.

3.2 Paramfit

Paramfit [12] is a part of the AmberTools software package [13]. The Amber force field is similar to the CHARMM force field described in equation 2.2. One of the major differences to AFMM is that it uses a merit function based on fitting energies of different conformations instead of NMA results. The merit function in Paramfit is defined as:

$$f(N, E_{QM}, K) = \sum_{i=1}^{N} [(E_{MM}(i) - E_{QM}(i) + K)^2] \tag{3.2}$$

In the merit function (equation 3.2) MM-calculated energy E_{MM} is compared to the QM-calculated energy E_{QM} for N conformations. K is an offset constant that allows minimization to 0. Instead of fitting the energy it is also possible to fit the forces (first derivatives of the energy). This approach avoids expensive calculations of vibrational frequencies. [12]

Paramfit is a hybrid algorithm: It combines a genetic algorithm with a simplex algorithm. An initial population is created and genetic operators are applied to a certain set of the population members in each generation. In particular two crossover operators are used:

Intermediate recombination The child parameters are randomly chosen within the range of the parent parameters respectively for all parameters in the parameter vector.

Linear (single-point) crossover The parameter vector of the parents is split at a random point and the all parameters after the split are swapped.

If the best fitness remains unchanged, then a simplex algorithm with weak convergence is run. This approach is shown to find a global optimum in the Rastrigin [14] function up to 15 dimensions much faster than the genetic algorithm alone. [12]

CHAPTER 4

Genetic algorithms

Historically, there have been multiple approaches for what is nowadays known as evolutionary computation. The concept of **evolution strategies** was explored by Rechenberg [15] since the 60s while the concept of **genetic algorithms** was developed by Holland in the 70s [16][17].

Similarly to artificial neural networks, which are inspired by biological neural networks [18], genetic algorithms (GA) are inspired by genetic mutations that occur in nature [16]. Nature successfully uses genetic mutations to solve complex problems. For example, somatic hypermutation [19] is a process where the rate of point mutations in antibody V-regions becomes 10^6 higher than in other genes. This allows the immune system to protect against previously unknown (e.g. viral) threats. Similarly, viruses adapt to outsmart the immune system and reproduce more efficiently. The idea behind GA is to replicate the basic mechanisms of evolution in order to solve problems which are difficult to tackle by traditional methods.

4.1 Advantages and limitations

Supervised machine learning algorithms like neural network or support vector machines have a major disadvantage: They require significant amounts of data in order to create a useful model, a process which is known as training a model. They are not suitable for use case where such data is not available or where creating it would be disproportionally expensive.

While calculus-based methods are capable of finding local optima by either setting the gradient of the objective function to 0 and solving sets of equations, or by using local gradient methods like steepest descent, they also have significant disadvantages. The first disadvantage is that local methods only search the neighborhood and can miss better results which are further away from the starting point. The other disadvantage is that such methods heavily rely on the existence of a derivative of the objective function. Objective functions for real world problems often do not have a derivative and they lack the smoothness of simple mathematical functions. The gradient often has to be approximated for such methods. [20]

This advantage and other major advantages over numeric algorithms were summarized by Wirsansky [21, Ch.1]:

- **Global optimization**: Many traditional optimization algorithms only search the area around a starting point and can get stuck in local optima. GA on the other hand work with many solution candidates. They can therefore cover a large search space and are robust towards local minima as long as the population is diverse enough.
- **Handling problems with a complex mathematical representation**: GA only require a fitness function to evaluate possible solutions and do not rely on additional information about the search space like derivatives.
- **Handling problems that lack mathematical representation**: The fitness function does not require a mathematical representation. It is, for example, possible to use subjective user-determined fitness values.
- **Resilience to noise**: GA can be even used for problems where the output of the fitness function with the same input parameters is non-deterministic, for example sensor data or user input.
- **Support for parallelism and distributed processing**: Fitness evaluations, mutations, and crossovers between solution candidates can be done in parallel on different members of the population. This resolves many of the parallelization issues that occur with many algorithms.
- **Suitability for continuous learning**: GA can continue working even when the environment changes and previous populations can be used to continue the computation.

However, there are also limitations to GA which should be considered [21, Ch.1]:

- **The need to create special definitions**: Creating a proper representation of the problem in terms of chromosome and fitness function can be challenging. Selection, crossover, and mutation operators also need to be defined depending on the representation.
- **Hyperparameter tuning**: Similarly to other optimization methods, GA require suitable hyperparameters (e.g. population size or mutation rate) to find good solutions in a reasonable time.

- **Computationally intensive**: GA operate on multiple solution candidates instead of just one. This makes the computation more repetitive.
- **Premature convergence**: A low population diversity, in other words, when the similarity between population members is too high, the population can converge to the fittest member and get stuck in a local minimum.
- **No guaranteed solution**: There is no guarantee that the global optimum will be found.

It should be noted that these advantages and limitations are merely a generalized summary. Some optimization algorithms and GA might address the aforementioned issues, but a detailed analysis of the large variety of algorithms is out of the scope of this thesis. For choosing optimization methods it is suggested to take a look at domain-specific solutions for optimization problems, which is done here for the topic of force field parametrization in chapter 3.

4.2 Terminology

In microbiology a **chromosome** is a string of DNA that encodes certain features of an organism. It consist of chains of the molecules adenine, cytosine, guanine, and thymine, which can be written in letters, e.g. `ACGTCTT`. In the context of genetic algorithms, the most basic type of chromosome consists of a binary chain, e.g. `0100110`. The single elements that the chromosome is composed of are called **allele**. Events where bases are inserted, removed, or replaced within such a string are known as **mutations**. The **fitness** traditionally describes the reproductive effectiveness of an organism, but in the case of GA it can be any function computed from the string that describes the (actual or approximated) quality of the solution that the chromosome represents. The **population** is the collection of chromosomes within a GA. Most animals have cells with multiple chromosomes, but in this case we will assume that only one chromosome is used per population member. Each population member is a set of parameters, or a solution candidate, for the problem that we are trying to solve. [17][16]

A genetic algorithm creates an **initial population** (generation 0) and follows a sequence of steps that lead to a new population, ideally with an improved mean fitness. This process is repeated until a stop condition is triggered. The solution candidate with the best fitness in the final generation is the result of the computation.

4.3 Basic algorithm

There are many adaptions and variations of the GA which differ in detail, but the fundamental concepts are similar. One of the earliest algorithmic models [16] can be seen in figure 4.1. Each chromosome is selected from the population for reproduction with a probability proportional to its fitness value. Operators are applied to the copy and the modified copy then replaces a random element in the population.

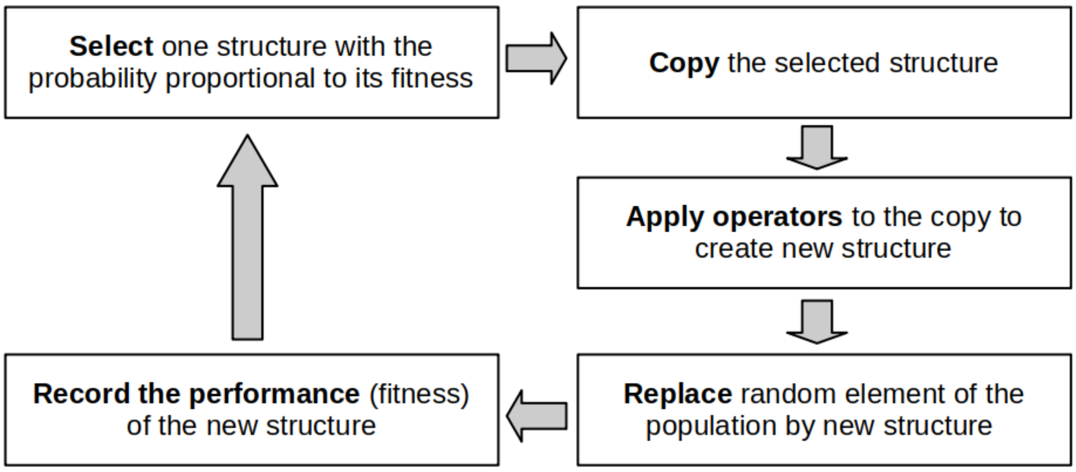

Select one structure with the probability proportional to its fitness
Copy the selected structure
Apply operators to the copy to create new structure
Replace random element of the population by new structure
Record the performance (fitness) of the new structure

Figure 4.1: Basic GA cycle as described by Holland [16, Ch.6].

Based on this foundation Goldberg investigated the application of GA in search, optimization and machine learning [20]. The canonical GA as described by Goldberg [20] includes three basic components:

- **Selection/Reproduction**: Candidate solutions from the population are copied into the *mating pool* with a probability according to their fitness. A higher fitness indicates a higher probability of reproduction.
- **Crossover**: Two of the candidate solutions from the mating pool are matched with each other, split at a random point, and all the values after the split swapped, therefore creating new candidate solutions for the next generation.
- **Mutation**: A secondary mechanism of genetic adaption that causes bit flips (e.g. 0 to 1 or 1 to 0) once in a while.

Instead of choosing the population members individually like in figure 4.1, the whole population is evolved into a new generation. The process of selection, crossover, and mutation happens between every generation, see figure 4.2.

4.4 Encoding

Before a problem can be solved using GA it needs to be translated into the GA language. Chahar et al [22] lists six methods of encoding: **Binary, octal, hexadecimal, permutation, value,** and **tree encoding**.

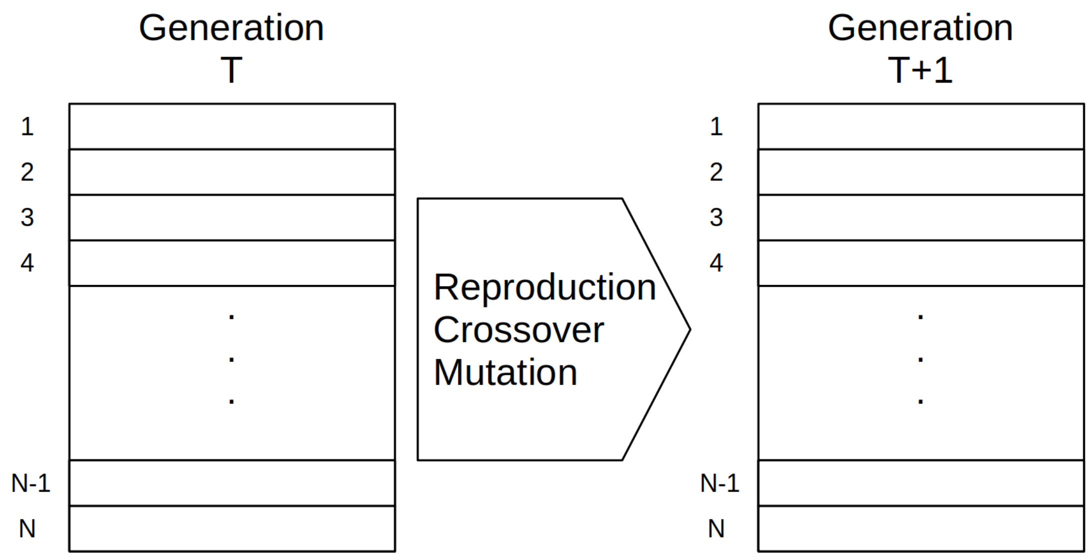


Figure 4.2: Model of a simple genetic algorithm as defined by Goldberg [20, Ch.3]. The population of generation `T` evolves to the population of generation `T+1` by following the process of reproduction, crossover, and mutation. The population size stays the same throughout the generations.

In most literature crossover and mutation operators are defined based on **binary encoding**, where the chromosomes are strings composed of 0 and 1 characters. Alternatively, **octal** or **hexadecimal encodings** can be used instead. In **permutation encoding** each allele describes the position of an object in an ordering problem. Genetic programming, a method of using genetic algorithms to generate sequential program code, uses **tree encoding** to encode the program flow and function calls. Finally, **value encoding** is used in cases where the previous encodings are not suitable. The values can be, for example, integer numbers, characters, or real numbers. GA based on real number encodings are referred to as **real-valued genetic algorithms** (RGA) and require modified crossover and mutation operators. [22][21]

4.5 Fitness

One of the challenges of GA is finding the appropriate fitness function, or objective function, that describes the *goodness* of a solution candidate (chromosome). From the point of view of a GA the fitness function is a black box and its implementation is domain-dependent. Although multi-objective genetic algorithms (MOGA) [23] exist, most literature refers to fitness functions for single-objective optimization.

Chromosome	**Fitness**
100110	5
110010	12
010000	9

Table 4.1: Example for fitness values.

4.6 Operators

Figure 4.2 describes the general generation-based GA, but there are many techniques to implement the single steps. Some of the most popular operators are described in this section.

Selection	**Crossover**	**Mutation**
Roulette wheel	Single-point	Bit flipping
Rank	Multi-point	Displacement
Tournament	Uniform	Inversion
Stochastic universal sampling		Scramble
Fitness scaling		

Table 4.2: Basic GA operator overview.

4.6.1 Selection / Reproduction

There is a variety of algorithms for selecting the chromosomes for reproduction. Five of the most important ones are listed below.

Roulette wheel selection The population members are mapped onto a wheel, which is then rotated randomly. The chance of selecting an element from the population is proportional to its fitness. The probability of selecting element i is $p = f_i/\overline{f}$ where f_i is the fitness of an element and $\overline{f}$ is the average fitness. This approach goes back to Holland and was further analyzed by Goldberg. [16][20, Ch.3][17, Ch.1][22]

Rank selection The population members are ranked by fitness and the probability of selecting an element is inversely proportional to its rank. [17][22]

Tournament selection There are different ways to implement tournament selection, but it is usually based on selecting candidate pairs and choosing the candidate with the higher fitness for reproduction. [17][22]

Stochastic universal sampling Unlike the roulette wheel selection, where the wheel is turned multiple times, the stochastic universal sampling has selection points uniformly distributed across the wheel and only requires one turn to choose all elements for reproduction. [21][22]

Fitness scaling This method is similar to the roulette wheel selection, but the fitness is not used directly to calculate the probability. Instead, a function is applied to scale the fitness value. [21]

4.6.2 Crossover

In the second step the selected chromosomes are paired up to create offspring for the next generation.

Single-point crossover The original single-point crossover as defined by Holland, works in three steps: First, two chromosomes with the length l are selected from the current population. Then a random number from 1 to $l-1$ is chosen. Finally, the alleles right to the position are swapped and two new chromosomes are created. [16][17][21]

If we take the first two chromosomes from table 4.1 and choose $l = 2$ as the splitting point it would lead to the exchange shown in table 4.3:

Chromosome before crossover	**Chromosome after crossover**
100110	100010
110010	110110

Table 4.3: Example for single-point crossover.

Multi-point crossover This approach works in analogous ways to the previous method, but with multiple splitting points. [17][21]

Uniform crossover The crossover splits are distributed uniformly (e.g. every second allele) over the chromosomes. [17][21]

4.6.3 Mutation

The mutation operator is a background operator that prevents the crossover operations from leading to premature convergence or getting trapped in local optima. [16, Ch.6]

Bit flipping This is a single operator where single bits in the chromosome are changed from 0 to 1 or from 1 to 0. This is one of the most common and basic operators for binary-coded genetic algorithms. [20, Ch.3][21][17]

Displacement A sequence of bits in the chromosome is moved to another position. [22]

Inversion A sequence of bits in the chromosome is reversed. [21][22]

Scramble A sequence of bits in the chromosome is randomly scrambled. [21][22]

4.7 Elitism

The old population can be fully replaced by the descendant generation, however, this brings the disadvantage that the best fitness or average fitness can get worse again. The concept of **elitism** is used to prevent this from happening: A number of chromosomes with the best fitness are retained for the next generation. Alternatively, the weakest population members are replaced by new population members. Another option is to replace a certain amount of randomly chosen chromosomes. Finally, it is also possible to use the tournament method where the fitness values of pairs of chromosomes are compared and the better one is accepted. [17][21]

4.8 Real-valued GA

Unlike discrete GA, real-valued GA (RGA) use continuous real values for the alleles instead of binary or other discrete symbols. Some operators, especially the mutation operators, need to be adapted for RGA. [17]. Two common operators are the **blend crossover** (BLX) and the **simulated binary crossover** (SBX). Note that unlike the previous operators, the RGA operators are applied *for each* value in the parents' chromosomes independently.

4.8.1 BLX Crossover

Two parent chromosomes are selected from the reproduction pool. Blend crossover is applied to each pair of parent chromosomes individually and the offspring value lies in the range [24][22][21]:

$$[parent_1 - \alpha(parent_2 - parent_1), parent_2 + \alpha(parent_2 - parent_1)] \tag{4.1}$$

α is a random number that lies in the interval between 0 and 1. The offspring interval for different α values is shown in figure 4.3. For $\alpha = 0$ the offspring value is chosen in the interval between the parent values.

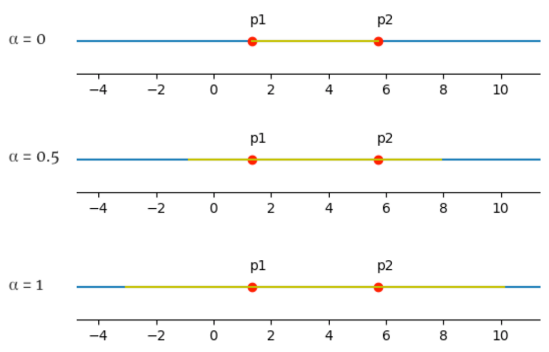


Figure 4.3: BLX crossover: The parents p1 and p2 produce offspring values in the yellow interval. The width of the interval depends on α. Figure from [21].

4.8.2 SBX Crossover

Similar to the binary crossover, the simulated binary crossover creates two offsprings. β is a (random) number larger than 0. The produced offspring is defined as [25][22][21]:

$$\begin{aligned} o1_i &= \frac{1}{2}[(1-\beta)x_i + (1+\beta)y_i] \\ o2_i &= \frac{1}{2}[(1+\beta)x_i + (1-\beta)y_i] \end{aligned} \tag{4.2}$$

Figure 4.4 shows the offspring value in dependence of β. For $\beta = 1$ the offspring replicates the parent value without any change.

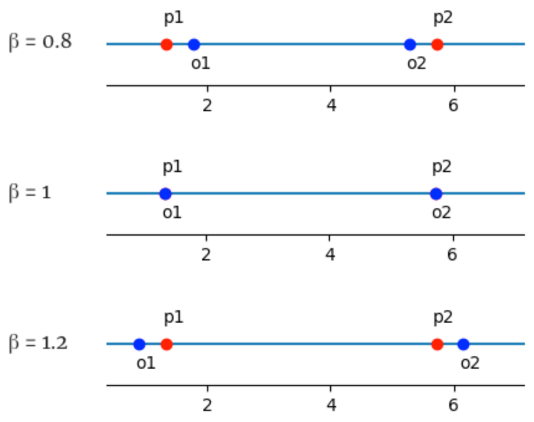


Figure 4.4: SBX crossover: Two offspring values o1 and o2 are created and β determines the similarity to the parent values p1 and p2. Figure from [21].

4.8.3 Mutations

Since bit flips are not suitable for mutating real values, different mutation operators are required. There are several publications on this topic (see [22]), but in this section only the two basic ones will be mentioned.

Replacement mutation Some values in the selected chromosome are replaced by random real values. In this approach the relationship to the original value is lost as it is replaced by a new randomly generated value [21].

Gaussian mutation A random number is generated based on a Gaussian distribution around the original value. In this approach the neighborhood of the original value can be explored [21].

4.9 No-Free-Lunch theorem

The variety of encoding methods and operators that can be used for genetic algorithms raises one question: Which ones are the best choice? Goldberg [20] did an early theoretical analysis of GA for optimization problems and Katoch et al [22] did a detailed review of papers involving genetic algorithms. It turns out that performance analysis of genetic algorithms tends to be very theoretical or problem-dependent.

The **No-Free-Lunch** theorem was analyzed and formalized by Wolpert and Macready [26]. It states that

> *[..] for both static and time-dependent optimization problems, the average performance of any pair of algorithms across all possible problems is identical.* [26]

Furthermore they claim that

> *[..] an algorithm's average performance is determined by how "aligned" it is with the underlying probability distribution over optimization problems on which it is run.* [26]

From this statement we can deduce that an optimization algorithm that works well on one problem should also work well on similarly structured problems, but that there is no single *best* algorithm.

CHAPTER 5

Problem analysis

The goal of this project is to use genetic algorithms to find good force constants K_* for the system- or molecule-dependent potential functions (equation 2.2) for use in molecular dynamics simulations of ionic liquids. Since most ionic liquids consist of a mixture of anions and cations in varying ratios, at least two types of molecules will be paired up for simulations. A good force field accurately replicates physical real-world behavior of such mixtures.

In our case the ultimate goal is to *use molecular mechanics to reproduce the observed IR spectrum from physical measurements.*

Unfortunately computing the IR spectrum directly is a time-consuming task which makes it unsuitable for use in fitness functions, because fitness functions need to be computed many times during the optimization process. Instead, we base the fitness function on comparing the normal modes.

5.1 Molecules and parameters

Optimal parameters for CHARMM [5] force fields and CLaP [27] force fields for 6 different molecules should be computed. The CLaP force fields have different values and need to be computed independently, but the principle is the same. The only difference is that the dihedral torsion angles can be negative and improper torsions are not used for optimization of CLaP force fields. Parameters for non-bonded interactions like van der Waals and electrostatic interactions are not optimized.

Table 5.1 shows the number of bond stretching, angle bending, and torsion parameters for each of the molecules. In total there are eight 8 to 45-dimensional optimization surfaces.

Molecule	Bonds	Angles	Dihedral t.	Improper t.	$\sum$
Im1	9	14	18	4	45
Im1H$^+$	8	12	17	4	41
Im1H$^+$ (CLaP)	8	12	17	0	37
OAc$^-$	3	4	1	1	9
OAc$^-$ (CLaP)	3	4	1	0	8
HOAc	5	6	4	1	16
OTf$^-$ (CLaP)	3	4	1	0	8
Im21$^+$ (CLaP)	9	14	19	0	42

Table 5.1: Number of force field parameters per type and total sum of parameters.

5.2 Fitness function

The fitness function, or objective function, represents the quality of a parameter set. The goal is to maximize or minimize this function by finding suitable parameters. Our fitness function (eq. 5.1) is based on comparing values from molecular mechanics-computed NMA (see section 2.3 for an explanation) with results from quantum mechanically precomputed NMA. A small error is desirable, which makes it a minimization task with the theoretical optimum at 0. The computation is the same as in AFMM, see section 3.1. In general, the sum of squared differences of vibrational frequencies is calculated and normalized by a quantum factor of 0.957 [5]. The Hungarian method [28] is used to match the vibrational frequencies to their respective eigenvectors.

$$fitness(mdfreq, mdX, mdY, mdZ, qmfreq, qmX, qmY, qmZ) \tag{5.1}$$

In the following example QM-calculated vibrational frequencies for OAc$^-$ are shown on the left (normalized with quantum factor 0.957) while MD vibrational frequencies for optimized parameters are shown on the right.

```
  38.0832      :   37.1461
 399.6353      :  399.0209
 578.1499      :  571.3312
 585.1879      :  586.6166
 818.8192      :  830.4762
 922.3170      :  914.9241
 961.9735      :  948.7063
1225.5265      : 1256.8892
1301.2723      : 1321.7307
1368.0702      : 1339.7813
1379.1796      : 1348.1398
1668.8337      : 1680.4878
2848.6474      : 2830.5436
```

```
2910.3859          :  2930.0955
2933.2282          :  2930.8579
```

The fitness (with normalized QM values) in this example is 22.39.

5.3 Fitness landscape

In order to choose an appropriate optimization function, it is important to understand what the optimization landscape looks like. Dimension reduction techniques like principal component analysis are commonly use to visualize higher dimensional functions, but are difficult to interpret intuitively. **Axis-parallel** plots like in figure 5.1 can be used to visualize high-dimensional functions with simple curves while preserving their geometry and providing an intuitive understanding [29].

A typical function landscape is represented by the **Himmelblau's function** [30]. It is a multimodal function that can be used for testing purposes. It has two input parameters (x and y), one output value ($z = f(x, y)$), and four global optima. Himmelblau's function is defined as:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \tag{5.2}$$

Figure 5.1 shows an example of how axis-parallel slicing works. It provides an understanding of the surface shape by splitting it from one plot to one plot *for each dimension*. Higher dimensional functions cannot be visualized easily, but it is possible to create axis-parallel 1D slices [29]. For example, the 9-dimensional fitness landscape for the OAc^- (acetate) anion can be seen in figure 5.2. We can assume that there are several local minima. However, there are a few areas with a smoother geometry that could profit from traditional algorithms.

Slicing answers question such as: What does the surface look like? Are there minima, discontinuities, convexities? Is a trend towards a minimum visible? It also resembles a sensitivity analysis over the full optimization space, in other words, it answers the question which parameters are more sensitive to changes.

Based on these observations in figure 5.2 the following conclusions can be made:

1. The surface is non-convex.
2. There are several local minima on the 1D slices.
3. There are roughly parabolic, ascending, or flat shapes.
4. The first bond stretching parameter (dim 0) and the first angle torsion parameter (dim 3) are important.
5. Dimensions 1, 4, 5, and 8 seem to have little impact on the fitness.

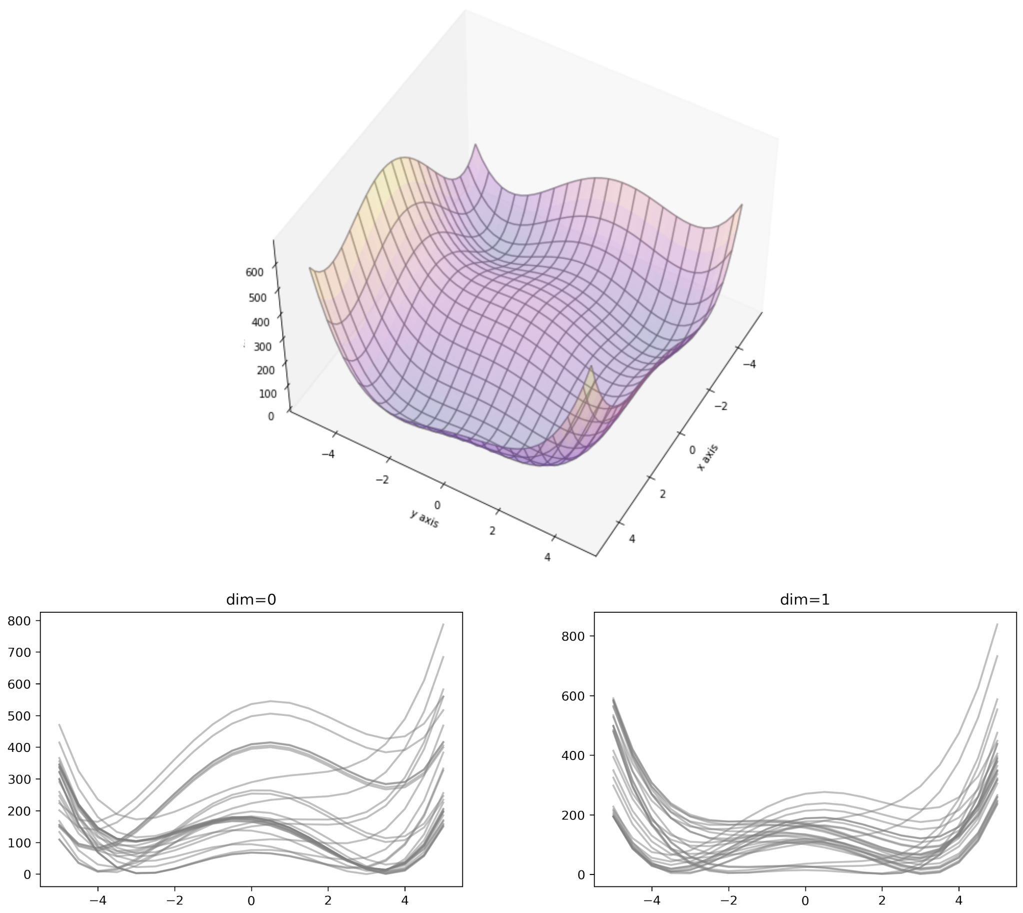


Figure 5.1: Full plot and axis-parallel plots of Himmelblau's function [30].

5.4 Challenges

The idea is to use a hybrid genetic algorithm, consisting of a real-valued genetic algorithm, for searching large areas, and a local neighborhood optimization algorithm for descent direction optimization. However, there are a few challenges involved with finding optimal parameters.

5.4.1 Curse of dimensionality

The Curse of Dimensionality describes a number of related problems that occur in math, statistics, and machine learning when dealing with high-dimensional data. One of the problems is **combinatorial explosion**. Sampling a 1-dimensional function $f(x)$ where $x \in [1, 100]$ and $x \in \mathbb{N}$ requires 100 samples. A 2-dimensional function with the same intervals for both parameters would require 10^4 samples, a 3-dimensional function would require 10^6 etc. Sampling the surface becomes a challenge: If one function evaluation

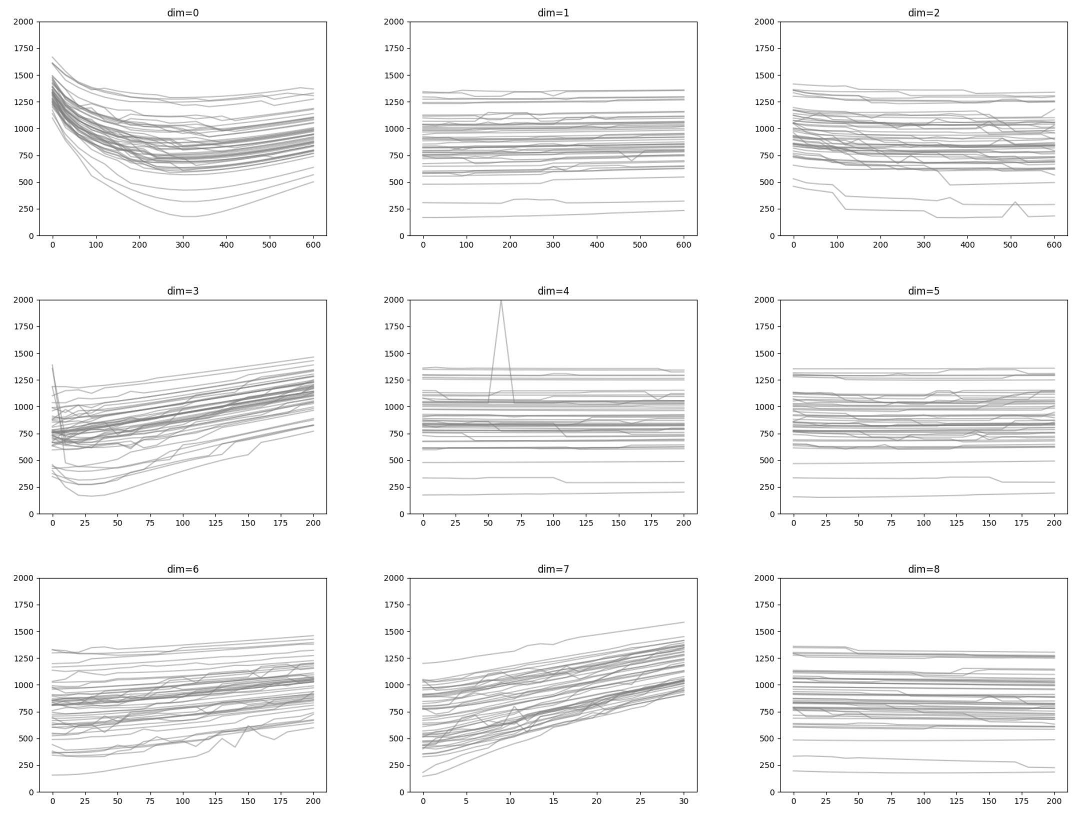


Figure 5.2: OAC Axis-parallel 1D plots: 50 slices with 21 samples for 9 dimensions. The top row represents varying bond stretching coefficients, dim 3 to dim 6 represent angle bending coefficients, dim 7 a dihedral torsion coefficient, and dim 8 the improper torsion coefficient. The spike in dimension 4 is a computation error.

takes 1 second, then creating 100 samples would take less than 2 minutes while sampling a 3D function in the same interval would take over 277 hours, or 11.57 days. This problem becomes even more severe for real-valued data.

Köppen [31] makes a few interesting observations about the curse of dimensionality by analyzing high dimensional cubes (hypercubes) and spheres (n-balls):

> *The distance of a randomly selected point in a hypercube of diagonal length 1 to one corner gets closer and closer to* $1/\sqrt{3}$ *as n increases.* [31]

Furthermore he concludes:

> *Two randomly selected points in a hypercube will have nearly the same distance for larger n. [..] An estimation of the volume occupied by a random set of points (basing e.g. the Monte Carlo method) is no more reliable.* [31]

We conclude that random uniform sampling is not suitable for high-dimensional input domains.

5.4.2 Sampling time

Calculating the full fitness function, including normal mode analysis and use of the Hungarian method, takes approximately 0.3 to 0.6 seconds depending on the molecule, force field type, and system usage on the working machine. 1 million computations would take approximately 83 to 167 hours.

System	Fitness computation time (seconds)
oac	0.52
hoac	0.29
im1	0.50
im1h	0.33
oac_clap	0.29
im1h_clap	0.33
im21_clap	0.34
OTf$^-$_clap	0.30

Table 5.2: Measured computation time for fitness.

5.4.3 Premature convergence

Premature convergence occurs when the algorithm gets trapped in a local minimum. The population members become so similar to each other that crossover operations do not lead to new candidates. This issue is tackled with different strategies: Mutation operators, the use of a second optimization algorithm, and diversity control, which is discussed in chapter 6.4.

CHAPTER 6

Algorithm design

The algorithm consists of two parts: a genetic algorithm, and a simple neighborhood optimization algorithm. Both algorithms should compensate each other's weaknesses. This approach is inspired by Paramfit [12].

6.1 Overview

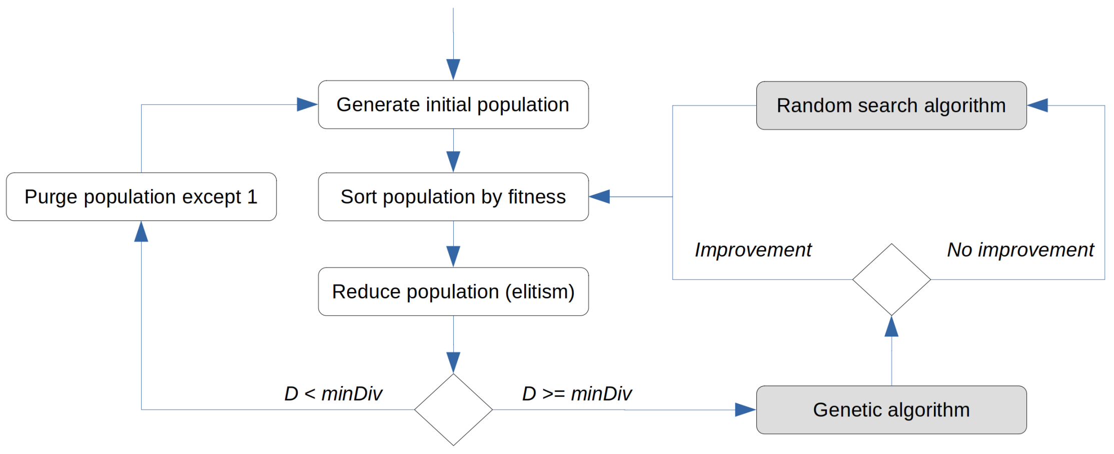

Generate initial population
Random search algorithm
Purge population except 1
Sort population by fitness
Improvement
No improvement
Reduce population (elitism)
D < minDiv
D >= minDiv
Genetic algorithm

Figure 6.1: Flowchart overview of the general algorithm

The general flow of the algorithm is shown in figure 6.1: After generating the initial population and sorting it by fitness, a genetic algorithm is used. If the genetic algorithm fails to achieve an improvement of the best fitness compared to the last generation, a random search algorithm is additionally used. if the population diversity becomes too low, a new population is created (see section 6.4). The population reduction step is required in order to keep the population size constant. A part of the implementation is shown in listing 6.1.

Listing 6.1: Hybrid algorithm in Python

```
self.p.generateInitialPopulation(self.p.maxSize)
self.p.sortPopulation()
for i in range(self.generations):
    if(self.p.estimateDiversity()<self.minimumDiversity):
        self.p.cutPopulation(1)
        self.p.generateInitialPopulation(self.p.maxSize)
    self.geneticOptimization(self.go.rankSelection)
    if(i != 0 and self.bestFitness[-1]==self.bestFitness[-2]):
        self.randomOptimization()
```

6.2 Genetic algorithm

The genetic algorithm is the first core component. The flowchart in figure 6.2 provides a detailed view of the *Genetic algorithm* component in figure 6.1. It is a **real value coded genetic algorithm** (RGA) and the selection, crossover, and mutation operators are chosen accordingly.

Selection operators Implemented are **roulette wheel selection** and **rank selection** (see section 4.6.1) as well as uniform random selection. The rank selection method is used for all experiments due to its simplicity and balance between preferring better population members and giving new population members a chance. Additionally, the ranking (sorting) is already included in the algorithm.

Crossover operators Two RGA-specific operators, BLX (section 4.8.1) and SBX (section 4.8.2), as well as the uniform crossover (section 4.6.2) are implemented. The experiments in Chapter 10 show that all of them work, with BLX being preferable over SBX and SBX preferable over uniform crossover. The probability of choosing either of the operators is defined in the configuration file.

Mutation operators Mutations are based on adding values from a Gaussian distribution (section 4.6.3) and scaling them accordingly to the preconfigured parameter range for each dimension.

Elitism After the crossover and mutation operations all new solution candidates are added to the population, the population is sorted by fitness, and all members ranked below the maximum population size are cut off. This procedure is called elitism, see section 4.7.

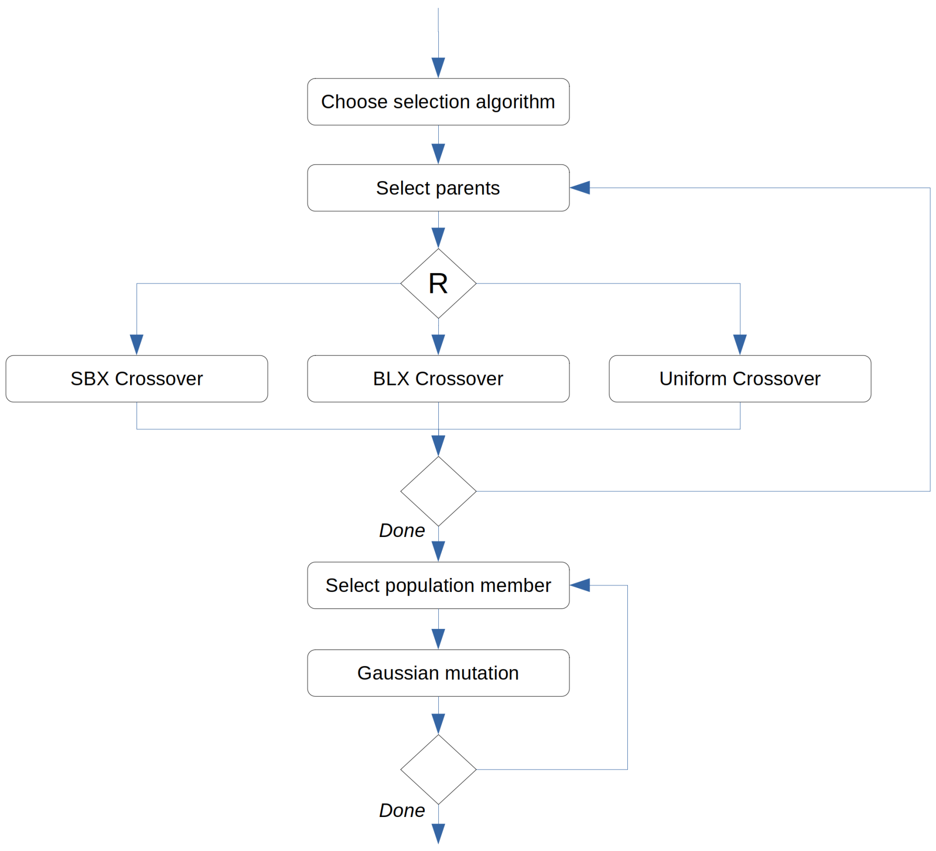


Figure 6.2: The genetic algorithm consists of a crossover stage and a mutation stage. The crossover method is chosen with a predefined probability while the mutation is computed by adding Gaussian random vectors.

6.3 Neighborhood optimization

Gradient-based methods are preferred for optimization problems where analytical gradient computation is available. Due to the chaotic nature of our function surfaces and relatively computationally intensive fitness evaluation, it seems reasonable to use an algorithm that does not rely on sampling and is robust enough to deal with randomness and varying numbers of dimensions. For this purpose we use a **random vector algorithm**.

6.3.1 Advantages and disadvantages

The random vector algorithm has a few advantages:

- Simple implementation and concept.
- Only one hyperparameter is required: the step size.

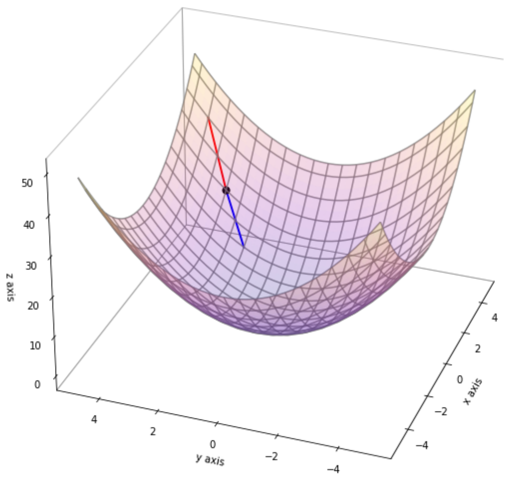


Figure 6.3: If a random direction shows an increasing slope (red), then the inverted direction is decreasing locally (blue) and vice versa.

- It works for any number of dimensions.
- No gradient calculation is required.
- Most function evaluations should lead to improvements unless minima are reached.

However, there are also disadvantages:

- The algorithm does not keep track of previous results which might be useful for optimization.
- It can get stuck in local minima like most neighborhood algorithms.
- Good step size can be difficult to determine: Too small step size leads to slow convergence and getting stuck in small local minima, too large step size lowers the chance of finding a descent direction.

The first two disadvantages are mitigated by the use of a genetic algorithm and the step size issue can be partially resolved by changing the step size after improvement failures.

6.3.2 Algorithm

The idea is as follows: Let $f(p)$ be a continuous and differentiable function. Given $||\Delta d|| \to 0$ and $||\frac{\partial f}{\partial p}(p_0)|| \neq 0$: if Δd is an ascent direction then $-\Delta d$ is always a descent direction and vice versa, see figure 6.3 for a graphical representation. If Δd is neither a descent nor an ascent direction, a new Δd can be computed in hope to find a descent direction.

Unfortunately we cannot rely on this assumption for our optimization problem because $||\Delta d|| \to 0$ is not a practical choice. Additionally, the fitness function is most likely not a continuous function. There is no guarantee that this algorithm works in all cases, but it can assist the genetic algorithm and prevent premature convergence by locally optimizing solution candidates.

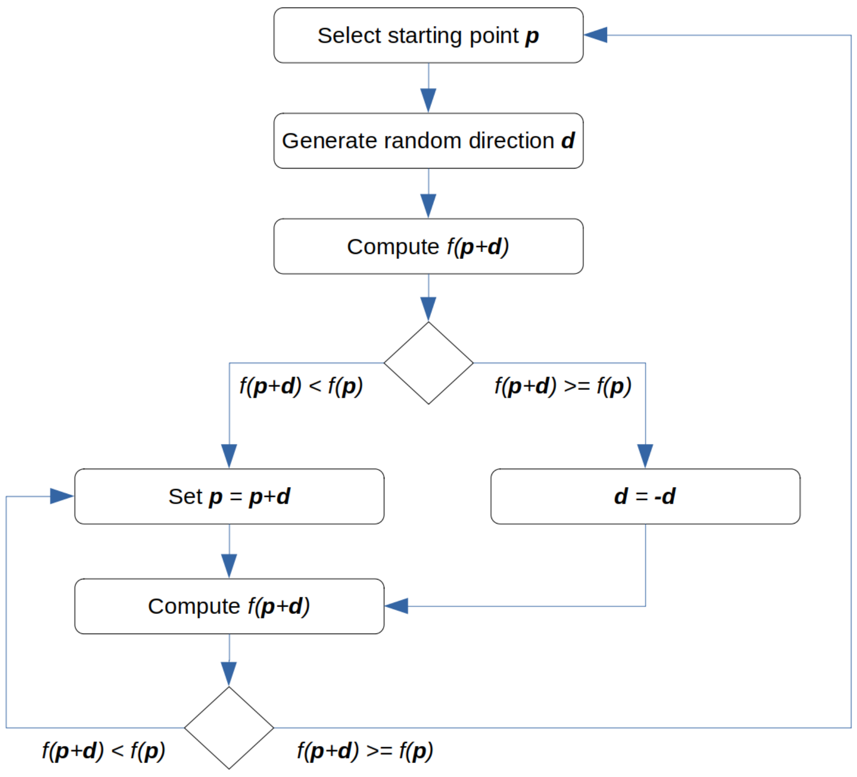


Figure 6.4: Local neighborhood optimization by choosing random directions. Convergence checks are omitted in this chart.

A flowchart of the simplified algorithm is shown in figure 6.4. Intuitively it can be understood like this: Select a parameter set from the population (using the rank selection operator from 4.6.1). Move into a random direction and check if the fitness function is improved. If not, take the opposite direction. In the ideal case, about 50% of such function evaluations should directly lead to improved parameters. If the fitness function improves, the same vector is added over and over again until the fitness gets worse again. In practice, additional criteria are needed, for example a minimum improvement per step.

6.3.3 Experiments

Figure 6.5 shows that multiple minima can be found from the same starting point. This is due to the fact that the algorithm is non-deterministic. This property is especially useful when multiple starting points from a population of solution candidates are available.

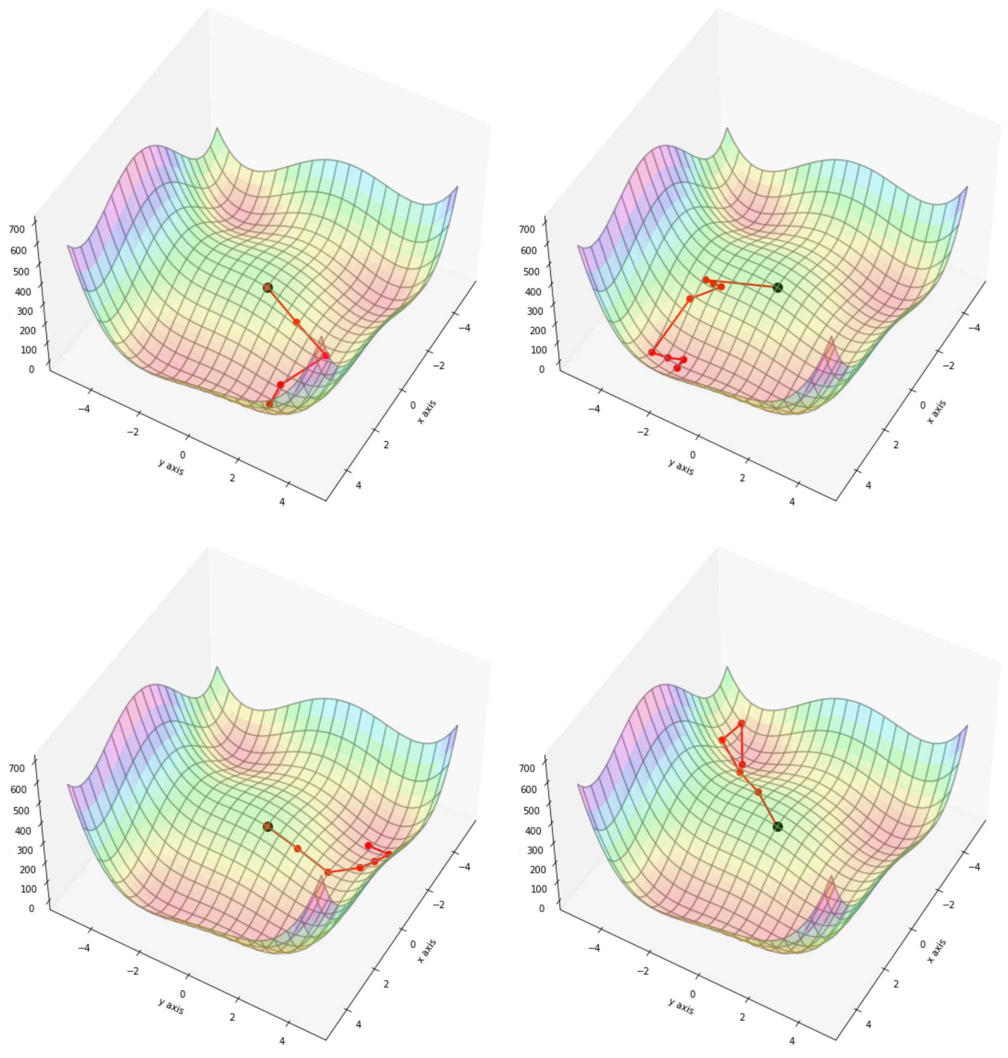


Figure 6.5: Four runs of optimizing Himmelblau's function [30] with the same initial position $x = 0$ and $y = 0$, but four different results.

6.3.4 Convergence measurements

The random vector algorithm works for higher dimensional problems. Figure 6.6 shows how many function evaluations are required to find a parameter set with a fitness below

0.1 for the 2-, 8- and 50-dimension square function in 10000 trials. The N-dimensional square function is defined as: $g(\vec{x}) = \sum_{i=0}^{N} x_i^2$

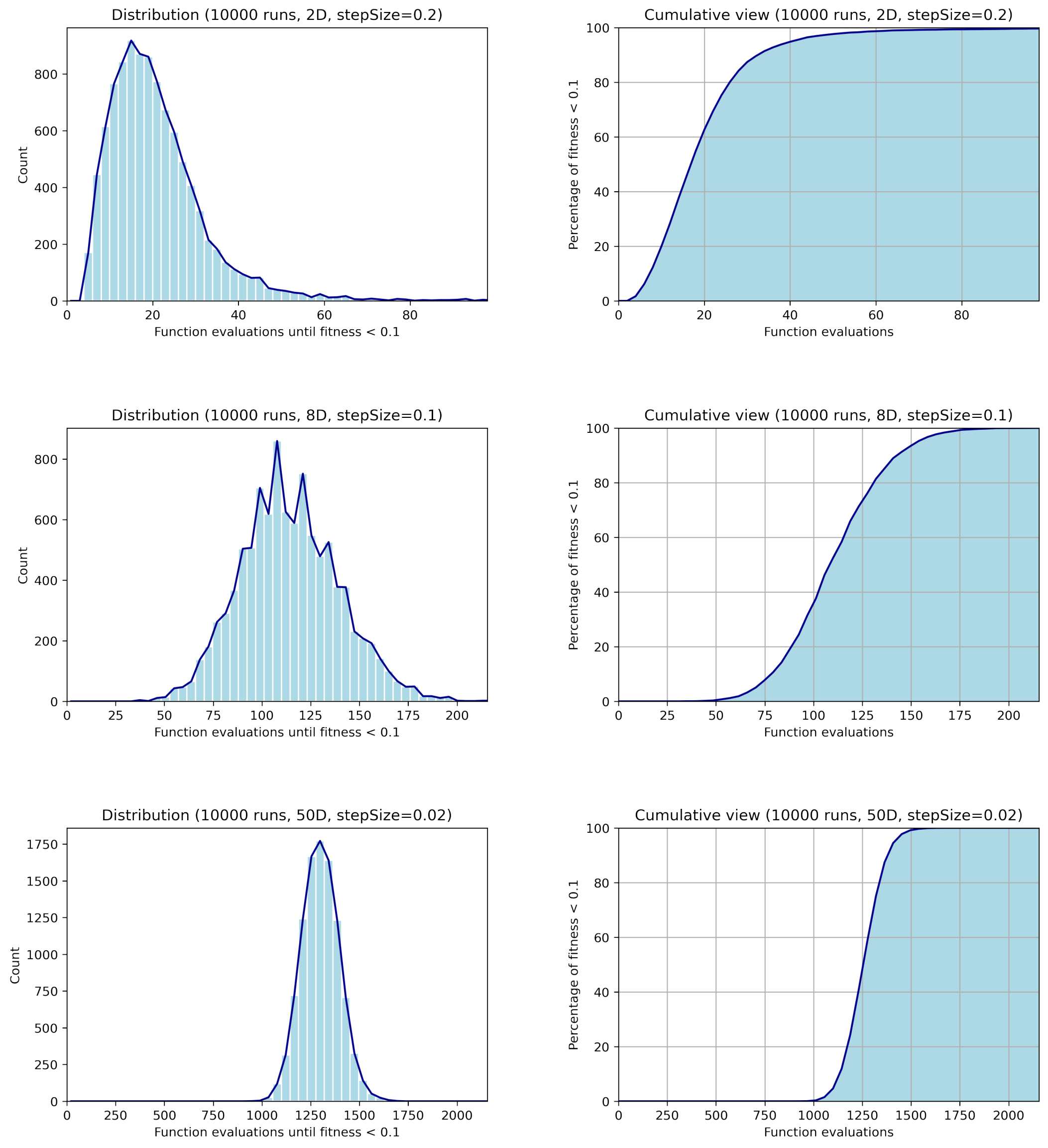
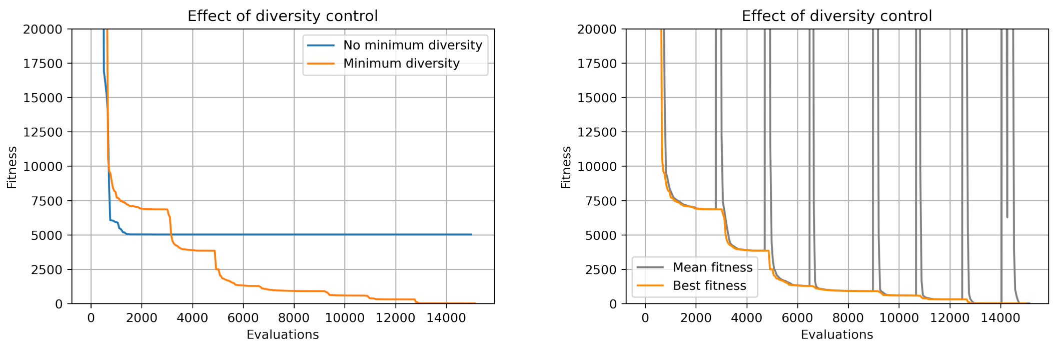


Figure 6.6: Convergence of the random vector algorithm. The algorithm was run 10000 times until the fitness is below 0.1 for the N = 2, 8, 50 square function. A smaller step size is preferable for larger N.

6.4 Diversity control

In theory, the best fitness value should improve with time, but in the case of premature convergence (section 5.4.3) the genetic algorithm wastes computation time with little, if any, improvement after a first successful phase as shown in figure 6.7 (a). Restarting the algorithm helps to break the barrier, but requires manual input and it leads to loss of previous results. When the population diversity is too low, in other words, when the population members are too similar to each other, a new population from a random distribution can be created. Only one population member is conserved into the new population in order to retain the result of the preceding computation. Figure 6.7 (b) shows what happens to the best fitness and the mean fitness of the population during such resets: The mean fitness explodes and the best fitness stagnates for a short while before improvement continues. However, too many population resets would lead to stagnation of the best fitness, therefore it is important to define a **minimum diversity** criterion below which the reset is triggered.



(a) The pure genetic algorithm (blue) stops improving after a few hundred evaluations, but the same algorithm with diversity control (orange) keeps converging towards a solution.

(b) The orange curve reflects the best fitness value while the black curve represents the mean fitness value of the population. The mean fitness increases when a new population is created.

Figure 6.7: Premature convergence and diversity control.

The diversity of a population can be estimated by computing the variance for each dimension. However, this requires additional computational effort. It is not necessary to have a precise value. The population is estimated on a random sample of 20% of the population members and on three randomly selected dimensions:

Listing 6.2: Diversity estimation in Python

```
def estimateDiversity(self,sampleDim=3,sampleSize=0.2):
    numOfSamples = max(1,int(len(self.members)*sampleSize))
    msd = 0.0
    for i in range(sampleDim):
        randDim = np.random.randint(0,self.settings.dim)
        for s in range(numOfSamples):
            randSample1 = np.random.randint(0,numOfSamples)
            randSample2 = np.random.randint(0,numOfSamples)
            while(randSample2==randSample1):
                randSample2 = np.random.randint(0,numOfSamples)
            msd += (self.members[randSample1].values[randDim]-
                self.members[randSample2].values[randDim])**2
    msd /= (numOfSamples*sampleDim)
    return msd
```

The diversity estimation with reset condition is repeated at the beginning of each new generation.

CHAPTER 7
Implementation

FF_GenOpt was developed and tested on Debian with Linux kernel 4.9. It is written in Python and works with Python 3.5.3+. It consists of multiple modular components which can be adapted for other fitness functions and use cases.

7.1 Requirements

The program requires a Python interpreter with **numpy** and **matplotlib**. The fitness computation modules requires **scipy** and a CHARMM installation.

7.2 Architecture

The program consists of multiple files which have to be in the same directory.

FF_GenOpt.py contains the core module, genetic algorithm, random search algorithm, log management, population management, and parameter management.

FF_GenOpt_conf.py contains the configuration file management.

FF_GenOpt_fitness.py wraps the fitness function which depends on the file below.

FF_GenOpt_extern.py contains modified code from AFMM and Florian Jörg (from the Department of Computational Biological Chemistry at the University of Vienna) for calculating the fitness function.

7.3 Configuration

FF_GenOpt uses configuration files that can be created and edited in any editor. The configuration files contain the execution command for CHARMM, the path to input files and output files, the optimization algorithm settings, and the parameter names, type, range, and initial values. Similar to Python, comments can be added by using # in the beginning. A typical configuration file is shown below:

```
MDEXEC                        charmm
MDINP                         Molecules/oac/oac.inp
MDOUT                         /dev/shm/opt.log
QMOUT                         Molecules/oac/oac.log
PARAM_FILENAME                FF_GenOpt_results/oac/parameters.
    str
POPULATION_FILENAME           FF_GenOpt_results/oac/
    lastPopulation_oac.txt
LOG_FILENAME                  FF_GenOpt_results/oac/
    convergence_oac.log

GENERATIONS                   10000
POPULATION_SIZE               200
MUTATIONS_PER_GENERATION      100
STEP_SIZE                     0.05
CROSSOVER_BLX                 0.7
CROSSOVER_SBX                 0.25
CROSSOVER_UNIFORM             0.05
MINIMUM_IMPROVEMENT           0.0001
MINIMUM_DIVERSITY             0.0001

#---------------------------------------------------------------
#           name   type          min        max        val
#---------------------------------------------------------------
PARAMETER oacb1 bond 0 600 340
PARAMETER oacb2 bond 0 600 655.5162
PARAMETER oacb3 bond 0 600 316.83
PARAMETER oaca1 angle 0 200 33.0
PARAMETER oaca2 angle 0 200 79.9417
PARAMETER oaca3 angle 0 200 70.0
PARAMETER oaca4 angle 0 200 35.7
PARAMETER oacd1 dihedral 0 30 0.0
PARAMETER oaci1 improper 0 200 71.000
```

`MDINP` points at the CHARMM input file, which contains all commands for reading the force field parameters, computing the normal modes with vibran, and *must point at the same file as* `PARAM_FILENAME` (which is rewritten each time the fitness function is computed).

`MDOUT` contains the output of CHARMM and is only used to compute the fitness function.

QMOUT contains the output of Gaussian with the QM-calculated normal modes. This file should not change during the optimization.

POPULATION_FILENAME contains the current population with all parameters, fitness values, the vibrational frequencies, and a small CHARMM script for setting the best set of parameters (on top). This file is rewritten after each generation, which enables the user to stop the algorithm at any time and take the best results.

LOG_FILENAME contains the log file, see section 7.5. It can be used to track and analyze the behavior of the algorithm with the current settings.

POPULATION_SIZE defines the maximum population size. A large population covers a large parameter search space, but the initial convergence is usually slower. High-dimensional problems have larger search spaces and require larger populations than low-dimensional problems. If the population is too small, the population diversity will shrink quickly and computation effort will be wasted on generating new population members. If the population is too large, computation time will be wasted on optimizing a vast number of population members, especially in the beginning.

GENERATIONS is the maximum number of generations to be computed. This number is an upper limit and should not be too small to prevent premature computation stops. Note that computation time and convergence speed per generation can vary significantly and comparing generational progress is not always useful.

MUTATIONS_PER_GENERATION is a *suggestion* for the number of crossover and mutation operations per generation. It is also used to control the number of attempts of the random search algorithm. The actual number of mutations and accepted new population members will differ between each generation.

STEP_SIZE is used exclusively by the random search algorithm and defines the component-wise *maximum* step size when computing a new direction vector. If this value is too small, the search algorithm wastes a lot of computation time with small steps and might get stuck in a small local minimum. If this value is too large then the idea of finding local descending directions by using the negative vector will fail often. For smooth high-dimension problems a smaller step size is preferable (see figure 6.6). However, the force field fitting problem does not have such a smooth surface. A value of 0.05 seems to be a good compromise. In this case at most 5% of the valid search range for each parameter will be used. It should be noted that this particular parameter is important and should be optimized if the performance is insufficient.

CROSSOVER_BLX, CROSSOVER_SBX, and CROSSOVER_UNIFORM define the probability for choosing either of the crossover operators per mutation operation in the genetic algorithm. The sum of all probabilities *must* be 1.0 or the program will fail. BLX and SBX are also parametrized, but the settings are fixed in code.

MINIMUM_IMPROVEMENT is a parameter for the random search that defines the minimum relative fitness improvement per step. This parameter is not as important as others, but it prevents wasting computation time on practically irrelevant improvements. 0.01 means that the improvement must be at least 1%, but the value can be much smaller.

`MINIMUM_DIVERSITY` is a parameter that prevents the algorithm from getting stuck in fruitless computations after all population members converged to the same values. Instead of stopping the algorithm here, a new initial population is created but the population member with the best fitness is retained. This value can also be relatively small, e.g. around 10^{-5}.

Finally, the `PARAMETER` keyword is used to define the parameter names and parameter type as well as the minimum, maximum, and original values. Genetic algorithms are designed for searching wide parameter ranges, but narrowing the parameter range down will lead to faster results.

7.4 Population files

Population files are generated at the end of each new generation and consist of three parts: All population members with their respective fitness value, the QM vibrational frequencies (multiplied by the QM factor of 0.957) and MM vibrational frequencies in absolute values and as a ratio (for the candidate with the best fitness), and the CHARMM script for setting the parameters of the population member with the best fitness.

Example population file for OAc^-:

```
Generation 1244
22.39,[316.2, 47.47, 581.35, 33.73, 75.29, 18.96, 21.96, 0.16, 75.17]
22.65,[316.31, 43.64, 579.66, 33.47, 75.37, 18.79, 22.56, 0.15, 77.39]
...
31.35,[319.67, 72.37, 602.14, 35.75, 68.69, 16.71, 18.39, 0.16, 99.42]
31.41,[315.9, 81.56, 587.97, 35.69, 73.06, 13.74, 16.8, 0.06, 77.63]

   38.0832      :   37.1461
  399.6353      :  399.0209
  578.1499      :  571.3312
  585.1879      :  586.6166
  818.8192      :  830.4762
  922.3170      :  914.9241
  961.9735      :  948.7063
 1225.5265      : 1256.8892
 1301.2723      : 1321.7307
 1368.0702      : 1339.7813
 1379.1796      : 1348.1398
 1668.8337      : 1680.4878
 2848.6474      : 2830.5436
 2910.3859      : 2930.0955
 2933.2282        :  2930.8579

      0.9754
      0.9985
      0.9882
      1.0024
      1.0142
      0.9920
      0.9862
      1.0256
      1.0157
      0.9793
      0.9775
      1.0070
      0.9936
      1.0068
      0.9992

set  oacb1  316.20
set  oacb2  47.47
set  oacb3  581.35
set  oaca1  33.73
set  oaca2  75.29
set  oaca3  18.96
set  oaca4  21.96
set  oacd1  0.16
set  oaci1  75.17
```

7.5 Log files

When starting an optimization process, the parameter settings are written into the log. The log is in TSV format and can be parsed easily by Excel-like tools or programming languages for analysis and plotting. A new line with 11 columns is appended after each generation. Example log file:

```
Gen     BestF   MeanF   TEvals  GEvals  REvals  t        t/eval
15      61.53   94.68   3092    1883    1209    1558.78  0.50
16      48.35   86.30   3201    1991    1210    1617.76  0.51
17      41.80   83.07   3313    2102    1211    1679.63  0.51

Delta   Improv  Divers
0.00    0.00    582.28
13.19   0.00    2639.27
6.55    0.00    345.59
```

The `Gen` column shows the current generation number. `BestF` and `MeanF` represent best fitness and mean fitness of the respective generation. `GEvals` and `REvals` show the number of function evaluations triggered by the genetic algorithm and by the rest of the algorithm (e.g. random search). `TEvals` is the sum of both. `t` is the total passed computation time in seconds while `t/eval` is the average time per evaluation. `Delta` is the improvement since the last generation, `Improv` is the improvement per evaluation, and `Divers` is the estimated diversity of the population.

7.6 Running FF_GenOpt

After creating the configuration file and setting up CHARMM inputs, the program can be started with

```
python3 FF_GenOpt.py configuration_file.cfg
```

It can be stopped by killing the process or keeping Ctrl+C pressed. Multiple sessions can be run at the same time remotely via the Linux tool **tmux**.

7.7 Slicing

The axis-parallel curve visualizations (see figure 1 in appendix A) are created by a simple tool named GridSlicer. It is based on the first version FF_GenOpt and uses the same configuration files. However, it requires different files to avoid interference with the optimization process and requires additional parameters.

```
python3 GridSlicer.py im1.cfg im1/im1.inp /dev/shm/im1.log im1/
    im1.log im1/parameters.str im1/ 50 21
```

The parameters are: configuration file, CHARMM input file, CHARMM output file, log file, CHARMM force field parameters file, image output path, number of focus points (a point that all curves cross), number of computations per curve per dimension. The number of dimensions can be added as an additional parameter to cut down computation time for testing purposes. The output files are in PNG format.

CHAPTER 8

Use cases

This chapter shows the original parameters and the optimized parameters for each system, and a comparison of QM-calculated infrared spectra and MM-calculated infrared spectra. The visualization of axis-parallel slices of all problems that are discussed in this chapter can be found in appendix A.

8.1 OAc$^-$

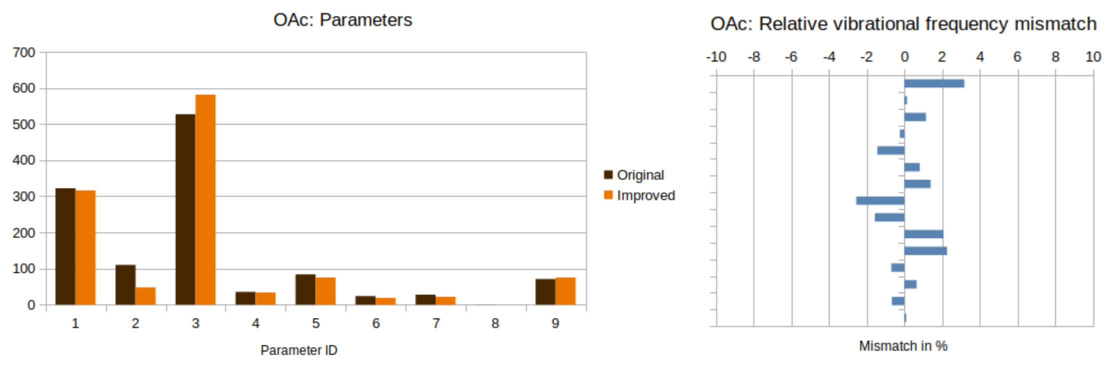


Figure 8.1: OAc$^-$: Comparison of original parameters with improved parameters (left) and relative vibrational frequency mismatch (right).

The optimized parameters shown in figure 8.1 (left) and table 8.1 are consistent with the axis parallel slices of the optimization surface in figure 5.2: The first bond stretching parameter appears to have its optimum around 320, the second has a slight upward slope and therefore a lower value, the third has a slight downward slope and therefore a higher value, and so on. Visually flat surfaces seem computationally difficult to optimize and lead to a greater variance between different optimization runs. Note that the parameter values are rounded and the fitness might be slightly different when recomputed using the rounded values.

	Original	Improved	Search range
Bond 1	322	316.09	[0,600]
Bond 2	110	47.659	[0,600]
Bond 3	527	581.295	[0,600]
Angle 1	35.5	33.717	[0,200]
Angle 2	83.7	75.232	[0,200]
Angle 3	23.8	18.968	[0,200]
Angle 4	27.7	21.948	[0,200]
Dihedral 1	0.42	0.161	[0,30]
Improper 1	71.0	75.141	[0,200]
Fitness	175.46	22.41	

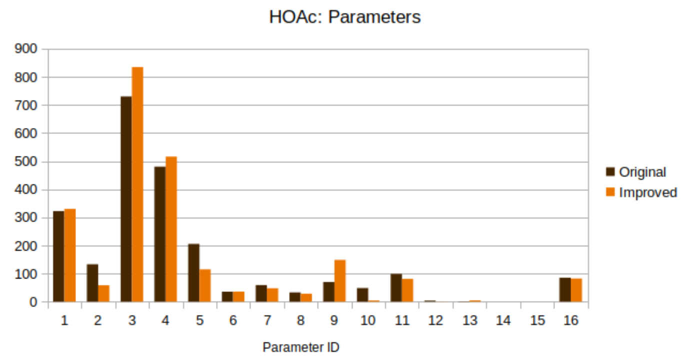
Table 8.1: OAc^-: Original parameters and improved parameters with respective fitness values.

The vibrational frequencies resulting from the best parameter set and the respective QM-calculated frequencies are shown below in listing 8.1. The MM-calculated frequencies are shown in the right column. Figure 8.1 (right) shows a visual representation of the vibrational frequency mismatch. It can be seen that the relative frequency mismatch is below 4% for all normal modes and the highest for the first frequency. In general, the fitness of 22.41 seems to be very close to the lower limit and most optimization runs result converge at a similar fitness value.

Listing 8.1: OAc Vibrational frequencies (QM-MM)

```
  38.0832      :   36.8705
 399.6353      :  399.0614
 578.1499      :  571.5083
 585.1879      :  586.5727
 818.8192      :  830.5424
 922.3170      :  914.7861
 961.9735      :  948.5750
1225.5265      : 1256.7690
1301.2723      : 1321.6670
1368.0702      : 1339.5965
1379.1796      : 1347.9676
1668.8337      : 1680.4147
2848.6474      : 2830.0919
2910.3859      : 2929.6162
2933.2282      : 2930.3799
```

8.2 HOAc



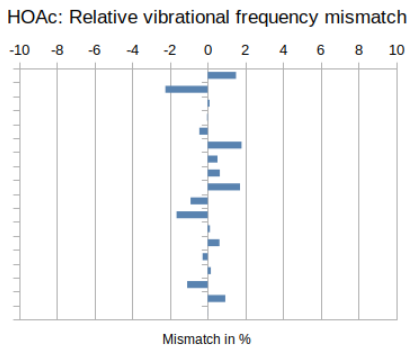


Figure 8.2: HOAc: Comparison of original parameters with improved parameters (left) and relative vibrational frequency mismatch (right).

The optimized parameters for HOAc in figure 8.2 and table 8.2 show significantly lower values for bond 2 and bond 5 parameters. The slices in appendix A show an upward slope for dihedral 3 (dim 13) and dihedral 4 (dim 14), which explains why the values are set to 0 for both, the original and the improved parameters. Overall, the old and new parameters have similarities for some parameters and differences for others.

	Original	Improved	Search range
Bond 1	322	330.22	[0,1000]
Bond 2	133	58.27	[0,1000]
Bond 3	730	834.38	[0,1000]
Bond 4	480	516.12	[0,1000]
Bond 5	205	114.92	[0,1000]
Angle 1	35.5	35.73	[0,200]
Angle 2	58.9	45.75	[0,200]
Angle 3	33	28.31	[0,200]
Angle 4	70	148.59	[0,200]
Angle 5	48.5	4.23	[0,200]
Angle 6	98.8	80.92	[0,200]
Dihedral 1	3.76	0.46	[0,30]
Dihedral 2	1.11	4.58	[0,30]
Dihedral 3	0	0	[0,30]
Dihedral 4	0	0	[0,30]
Improper 1	85.0	82.51	[0,300]
Fitness	73.26	13.74	

Table 8.2: HOAc: Original parameters and improved parameters with respective fitness values.

The frequencies match very well with the QM-calculated frequencies in figure 8.2 (left) and listing 8.2 with a maximum error of slightly over 2% for the second frequency.

Listing 8.2: HOAc Vibrational frequencies (QM-MM)

```
  111.6989        :   110.0166
  406.7286        :   415.8370
  444.1641        :   443.7043
  569.2877        :   569.4372
  571.4980        :   573.9926
  826.9086        :   812.0284
  937.4898        :   932.5550
 1000.6678        :   994.1870
 1155.7921        :  1135.9413
 1251.8640        :  1263.1710
 1317.1358        :  1338.8099
 1380.3030        :  1378.6221
 1394.0976        :  1385.3141
 1797.3517        :  1802.1365
 2905.0544        :  2900.1159
 2969.2253        :  3001.3816
 3030.2812        :  3001.8650
 3602.3177        :  3602.4117
```

8.3 Im1

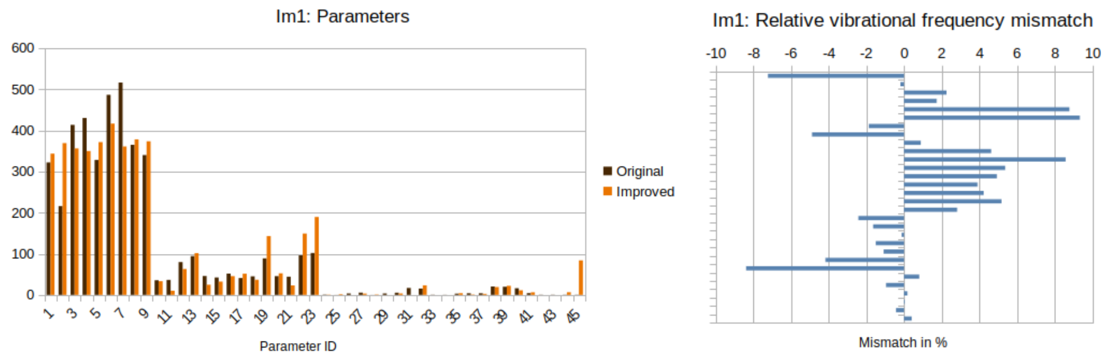


Figure 8.3: Im1: Comparison of original parameters with improved parameters (left) and relative vibrational frequency mismatch (right).

The potential function for Im1 has a 45-dimensional optimization surface which makes parameter fitting more difficult compared to the previous molecules. Figure 8.3 and table 8.3 show that some bond stretching parameters have been significantly decreased while angle parameters have been increased. The last improper torsion value has been increased from 0.45 to 83.845, which is significant. The slices for Im1 in appendix A show that changing this (last) parameter does not change the fitness value much for *most* samples, but there are some parameter sets where more structured changes of the fitness value can be seen.

	Original	Improved	Search range
Bond 1	322.00	343.396	[0,1000]
Bond 2	215.80	369.019	[0,1000]
Bond 3	413.37	356.072	[0,1000]
Bond 4	430.13	349.758	[0,1000]
Bond 5	328.03	371.493	[0,1000]
Bond 6	486.48	416.364	[0,1000]
Bond 7	516.27	360.815	[0,1000]
Bond 8	365.00	378.147	[0,1000]
Bond 9	340.00	373.416	[0,1000]
Angle 1	35.50	33.208	[0,200]
Angle 2	36.42	10.226	[0,200]
Angle 3	79.78	63.008	[0,200]
Angle 4	94.37	101.483	[0,200]
Angle 5	45.91	24.667	[0,200]
Angle 6	42.22	32.37	[0,200]
Angle 7	51.72	32.37	[0,200]
Angle 8	40.89	51.427	[0,200]
Angle 9	45.01	36.776	[0,200]
Angle 10	88.69	142.845	[0,200]
Angle 11	45.44	52.32	[0,200]
Angle 12	44.18	23.031	[0,200]
Angle 13	96.13	149.095	[0,200]
Angle 14	101.77	189.359	[0,200]
Dihedral 1	0.86583	0.634	[0,30]
Dihedral 2	0.19824	1.762	[0,30]
Dihedral 3	3.40614	0.024	[0,30]
Dihedral 4	5.44233	3.182	[0,30]
Dihedral 5	0.15530	0.913	[0,30]
Dihedral 6	3.51991	0.002	[0,30]
Dihedral 7	5.21739	3.429	[0,30]
Dihedral 8	16.78064	0.006	[0,30]
Dihedral 9	15.37087	23.214	[0,30]
Dihedral 10	0.41036	0.016	[0,30]
Dihedral 11	0.39232	0.004	[0,30]
Dihedral 12	2.88761	4.077	[0,30]
Dihedral 13	3.92983	0.998	[0,30]

Dihedral 14	3.84807	2.532	[0,30]
Dihedral 15	20.41265	19.178	[0,30]
Dihedral 16	19.75138	22.083	[0,30]
Dihedral 17	16.20717	11.111	[0,30]
Dihedral 18	4.20182	6.261	[0,30]
Improper 1	0.500	0.089	[0,300]
Improper 2	0.500	0.031	[0,300]
Improper 3	0.500	6.394	[0,300]
Improper 4	0.450	83.845	[0,300]
Fitness	154.28	50.94	

Table 8.3: Original and optimized Im1 parameters

The relative frequency mismatch (figure 8.3 and listing 8.3) for most frequencies is larger than previously, but still below 10%.

Listing 8.3: Im1 Vibrational frequencies (QM-MM)

```
  74.8268      :   80.2302
 200.3856      :  200.7725
 332.3679      :  324.8592
 595.5067      :  585.1736
 644.5991      :  588.0034
 649.9238      :  589.2624
 690.0785      :  702.9609
 770.7495      :  808.3889
 822.2342      :  814.8789
 873.8681      :  833.4257
 996.7885      :  911.2631
1020.5653      :  965.7318
1036.3011      :  985.2235
1082.8758      : 1040.6027
1091.7161      : 1045.5295
1200.2467      : 1138.1372
1246.9484      : 1211.7878
1322.5561      : 1354.6137
1341.2945      : 1363.3152
1373.0568      : 1374.7832
1392.3141      : 1413.1595
1425.6743      : 1441.0874
1475.6246      : 1537.1807
1476.3619      : 1600.1332
2905.7963      : 2882.2210
```

```
2968.9022      :  2997.1765
3005.4686      :  2999.9593
3096.1133      :  3094.9264
3098.4115      :  3111.4760
3127.9878      :  3115.1930
```

8.4 Im1H$^+$

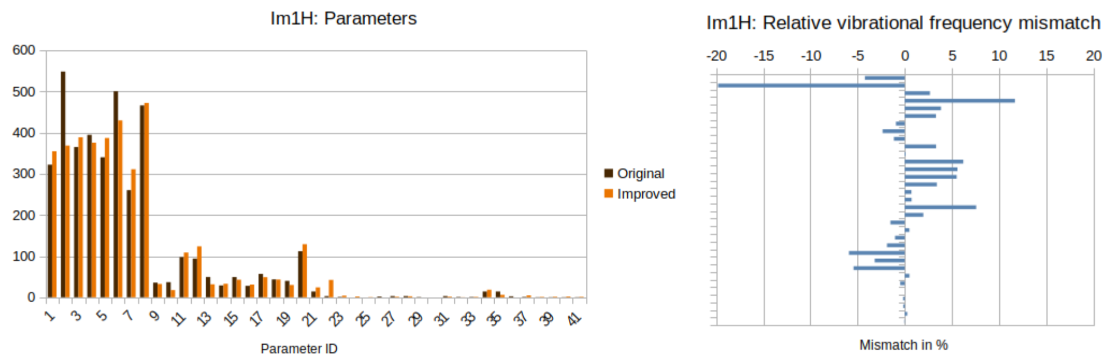


Figure 8.4: Im1H$^+$: Comparison of original parameters with improved parameters (left) and relative vibrational frequency mismatch (right).

The optimized parameters for Im1H$^+$ in figure 8.4 (left) and table 8.4 are similar to the original parameters in most cases. The second bond stretching parameter is significantly smaller than the original value.

	Original	Improved	Search range
Bond 1	322.00	354.699	[0,1000]
Bond 2	548.04	368.282	[0,1000]
Bond 3	365.00	388.481	[0,1000]
Bond 4	394.47	375.453	[0,1000]
Bond 5	340.00	386.76	[0,1000]
Bond 6	500.39	429.529	[0,1000]
Bond 7	260.13	310.873	[0,1000]
Bond 8	466.0	471.942	[0,1000]
Angle 1	35.50	32.219	[0,200]
Angle 2	36.73	17.502	[0,200]
Angle 3	97.52	108.592	[0,200]
Angle 4	94.00	123.711	[0,200]
Angle 5	49.24	31.471	[0,200]
Angle 6	28.72	32.754	[0,200]
Angle 7	49.11	42.493	[0,200]

Angle 8	27.82	30.863	[0,200]
Angle 9	56.76	48.836	[0,200]
Angle 10	54.35	43.022	[0,200]
Angle 11	39.84	29.777	[0,200]
Angle 12	111.97	128.771	[0,200]
Dihedral 1	14.00	23.942	[0,30]
Dihedral 2	2.90	42.165	[0,30]
Dihedral 3	1.10	3.85	[0,30]
Dihedral 4	0	2.263	[0,30]
Dihedral 5	0	0.772	[0,30]
Dihedral 6	2.00	0.022	[0,30]
Dihedral 7	3.00	1.457	[0,30]
Dihedral 8	3.00	2.509	[0,30]
Dihedral 9	0.80	0.059	[0,30]
Dihedral 10	0.00	0.021	[0,30]
Dihedral 11	3.00	1.641	[0,30]
Dihedral 12	1.00	0.375	[0,30]
Dihedral 13	0.90	1.089	[0,30]
Dihedral 14	14.00	18.33	[0,30]
Dihedral 15	14.00	5.913	[0,30]
Dihedral 16	2.40	0.055	[0,30]
Dihedral 17	1.10	4.594	[0,30]
Improper 1	0.45	1.104	[0,30]
Improper 2	0.50	1.386	[0,30]
Improper 3	0.50	1.903	[0,30]
Improper 4	0.45	1.339	[0,30]
Fitness	149.54	46.81	

Table 8.4: Original and optimized Im1H$^+$ parameters

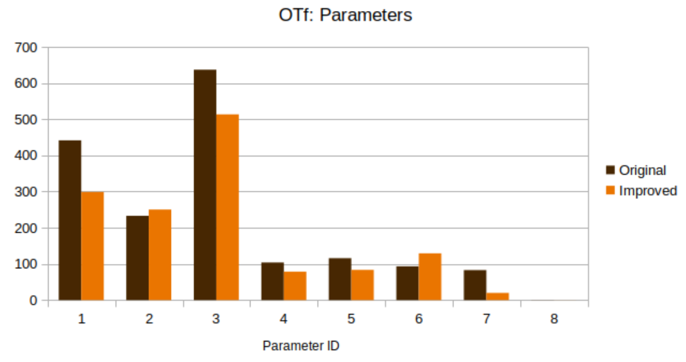
The second vibrational frequency (figure 8.4 (right) and listing 8.4) has a relatively large mismatch (about 20%). The remaining frequencies are reasonably close to the QM-calculated values.

Listing 8.4: Im1H Vibrational frequencies (QM-MM)

```
 87.3200     :  91.0086
230.5254     : 276.1720
338.2704     : 329.2314
591.8363     : 522.6837
608.6231     : 585.2435
631.8135     : 610.8808
659.3371     : 665.5023
```

722.5734	:	739.5322
800.5177	:	809.7607
844.1218	:	816.0263
887.4290	:	886.5838
976.0062	:	915.4072
1036.4925	:	978.5439
1052.4398	:	994.4853
1068.6999	:	1032.3307
1087.3481	:	1079.6320
1115.9416	:	1107.8688
1234.0748	:	1140.5745
1270.9672	:	1245.9459
1343.8767	:	1364.3376
1375.8430	:	1369.0574
1391.1119	:	1405.4976
1410.8491	:	1437.4823
1420.9776	:	1505.1693
1515.8409	:	1564.2093
1555.9109	:	1640.4832
2941.5852	:	2926.7697
3030.1237	:	3044.8509
3046.7247	:	3046.5491
3144.3648	:	3150.0198
3148.2939	:	3153.3148
3162.6898	:	3154.2529
3446.7015	:	3448.1426

8.5 OTf⁻ (CLaP)



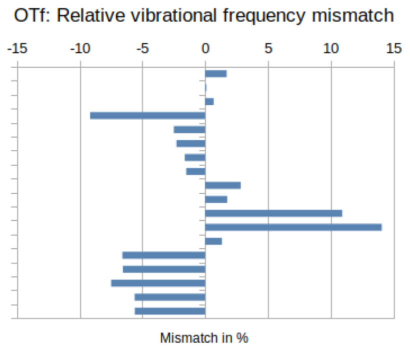


Figure 8.5: OTf⁻ (CLaP): Comparison of original parameters with improved parameters (left) and relative vibrational frequency mismatch (right).

Figure 8.5 (left) shows that the optimized CLaP force constants for Triflate are relatively similar to the original values, with the exception of the last angle bending parameter which is reduced from 82.88 to 19.87 (table 8.5).

	Original	Improved	Search range
Bond 1	441.6	298.72	[0,1000]
Bond 2	232.9	250.15	[0,1000]
Bond 3	636.6	513.20	[0,1000]
Angle 1	103.90	78.61	[0,200]
Angle 2	115.72	83.33	[0,200]
Angle 3	93.27	129.05	[0,200]
Angle 4	82.88	19.87	[0,200]
Dihedral 1	0.173	0.02	[-30,30]
Fitness	144.58	57.85	

Table 8.5: Original and optimized OTf⁻ (CLaP) parameters

The parameters for OTf⁻ were optimized by using weights for each frequency while computing the fitness value in the optimization process (as described in the AFMM [11] paper). This was done in order to ensure that the first vibrational frequency does not differ too much from the reference. However, the fitness value in table 8.5 refers to the unweighted fitness value (which is also used for every other example).

Listing 8.5: OTf (ClaP) Vibrational frequencies (QM-MM)

```
  44.1129       :    43.3560
 171.1195       :   170.9463
 172.1909       :   171.0111
 255.7675       :   279.2067
 305.5468       :   313.1689
 306.2727       :   313.2593
 463.1164       :   470.6204
 463.7793       :   470.7540
 520.1481       :   505.3826
 520.8335       :   511.6490
 574.4153       :   511.6545
 690.3114       :   593.1438
 921.5117       :   909.1162
1094.6458       :  1166.7832
1095.1101       :  1166.8466
1136.1638       :  1221.2345
1166.7584       :  1232.2111
1166.8864       :  1232.2451
```

8.6 OAc- (CLaP)

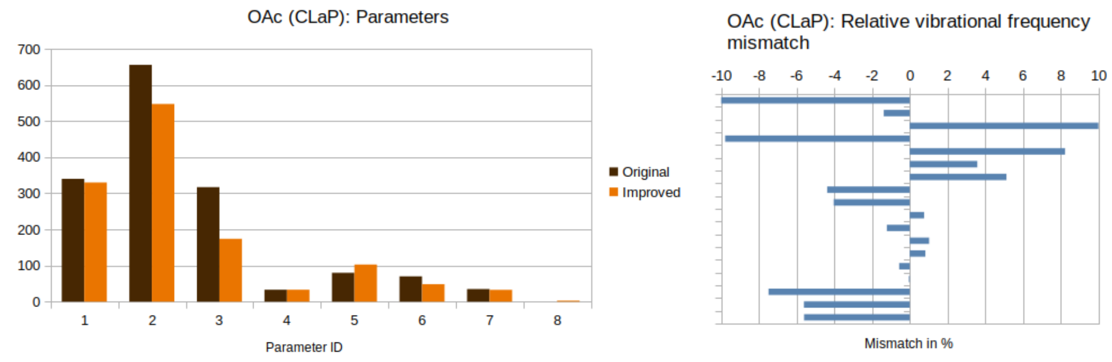


Figure 8.6: OAc- (CLaP): Comparison of original parameters with improved parameters (left) and relative vibrational frequency mismatch (right).

The optimized parameters are mostly similar to the original values, but the fitness does not improve as well as for the CHARMM force field parameter.

	Original	Improved	Search range
Bond 1	340.00	329.70	[0,1000]
Bond 2	655.5162	547.03	[0,1000]
Bond 3	316.83	174.03	[0,1000]
Angle 1	33.00	33.1	[0,180]
Angle 2	79.9417	102.68	[0,180]
Angle 3	70.00	48.22	[0,180]

Angle 4	35.70	32.58	[0,180]
Dihedral 1	0	2.79	[-30,30]
Fitness	147.61	49.08	

Table 8.6: Original and optimized OAc⁻ (CLaP) parameters

Unlike the CHARMM force field parameters, the CLaP parameters for OAc⁻ suffer from a huge frequency mismatch, especially for the first vibrational frequency. Note that this result is highly unstable. A slightly different set of parameters leads to a significantly worse fitness.

Listing 8.6: OAc (CLaP) Vibrational frequencies (QM-MM)

```
  38.0450       :  128.9063
 399.6349       :  402.4909
 578.1419       :  513.6252
 585.1890       :  643.3375
 818.8200       :  751.3653
 922.3173       :  889.8001
 961.9706       :  913.0012
1225.5175       : 1279.4620
1301.2734       : 1354.1282
1368.0746       : 1357.9662
1379.1687       : 1396.3218
1668.8343       : 1652.3428
2848.6564       : 2825.2567
2910.3768       : 2926.9125
2933.2753       : 2934.7594
```

8.7 Im1H$^+$ (CLaP)

The CLaP force field parameters for Im1H$^+$ are particularly hard to optimize. There are significant differences for almost all parameters and the fitness value is not satisfying.

	Original	Improved	Search range
Bond 1	340.00	186.28	[0,1000]
Bond 2	519.6525	128.10	[0,1000]
Bond 3	367.00	56.47	[0,1000]
Bond 4	426.7151	348.99	[0,1000]
Bond 5	367.00	102.31	[0,1000]
Bond 6	476.6817	522.72	[0,1000]
Bond 7	336.7752	414.93	[0,1000]
Bond 8	474.00	35.43	[0,1000]

Angle 1	33.00	111.36	[0,120]
Angle 2	34.9766	24.62	[0,120]
Angle 3	69.9530	93.12	[0,120]
Angle 4	69.9530	115.33	[0,120]
Angle 5	69.9530	120.06	[0,120]
Angle 6	35.00	71.65	[0,120]
Angle 7	69.9530	111.83	[0,120]
Angle 8	35.00	63.16	[0,120]
Angle 9	37.4749	67.53	[0,120]
Angle 10	34.9766	42.91	[0,120]
Angle 11	34.9766	64.79	[0,120]
Angle 12	69.9630	106.17	[0,120]
Dihedral 1	1.499	12.10	[-10,50]
Dihedral 2	1.499	44.14	[-10,50]
Dihedral 3	1.499	8.57	[-10,50]
Dihedral 4	0.062	5.09	[-10,50]
Dihedral 5	0	25.28	[-10,50]
Dihedral 6	5.371	13.57	[-10,50]
Dihedral 7	5.371	49.87	[-10,50]
Dihedral 8	1.499	50.63*	[-10,50]
Dihedral 9	1.499	-0.70	[-10,50]
Dihedral 10	1.499	-3.27	[-10,50]
Dihedral 11	2.324	5.45	[-10,50]
Dihedral 12	2.324	-6.67	[-10,50]
Dihedral 13	2.324	0.71	[-10,50]
Dihedral 14	5.371	1.45	[-10,50]
Dihedral 15	2.324	3.95	[-10,50]
Dihedral 16	2.324	2.89	[-10,50]
Dihedral 17	2.324	-0.02	[-10,50]
Fitness	1284.45	613.50	

Table 8.7: Original and optimized Im1H$^+$ (CLaP) parameters

The first frequency almost vanishes while the second frequency is almost twice as large compared to the reference value.

Listing 8.7: Im1H (CLaP) Vibrational frequencies (QM-MM)

```
 87.3525     :    0.0234
230.5271     :  451.9878
338.2728     :  491.7376
591.8376     :  583.2070
608.6232     :  611.5528
```

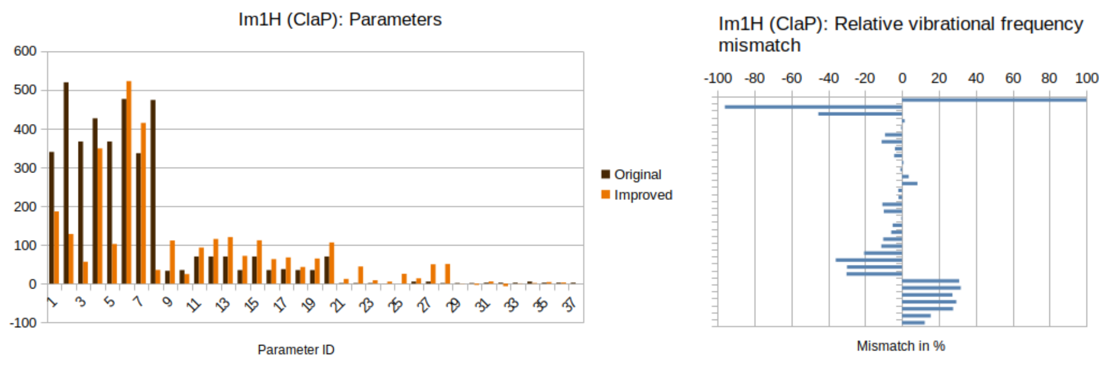
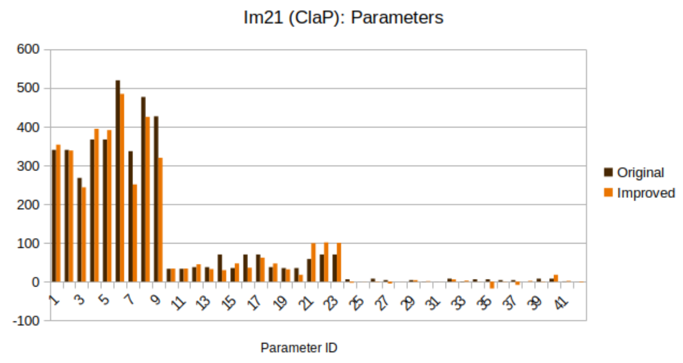


Figure 8.7: Im1H (CLaP)$^+$: Comparison of original parameters with improved parameters (left) and relative vibrational frequency mismatch (right).

631.8165	:	690.0446
659.3392	:	732.0716
722.5741	:	750.2822
800.5147	:	834.5622
844.1217	:	837.7344
887.4292	:	895.2278
976.0092	:	940.1690
1036.4947	:	949.7699
1052.4392	:	1074.1674
1068.7002	:	1088.8591
1087.3510	:	1203.1945
1115.9403	:	1226.5211
1234.0762	:	1238.6684
1270.9703	:	1335.3398
1343.8795	:	1422.3931
1375.8458	:	1515.8885
1391.1139	:	1547.3630
1410.8464	:	1700.9580
1420.9793	:	1931.8249
1515.8379	:	1968.2214
1555.9080	:	2024.1537
2941.5768	:	2028.1777
3030.1133	:	2065.1005
3046.7168	:	2212.5873
3144.3642	:	2218.0878
3148.2959	:	2275.0236
3162.6904	:	2670.4849
3446.7000	:	3020.7741

8.8 Im21$^+$ (CLaP)



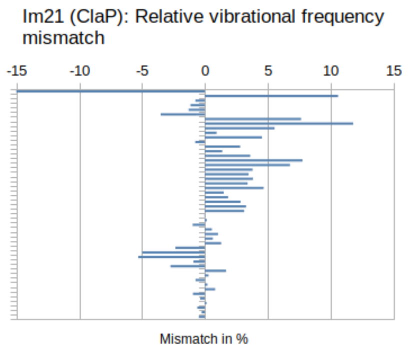


Figure 8.8: Im21$^+$: Comparison of original parameters with improved parameters (left) and relative vibrational frequency mismatch (right).

	Original	Improved	Search range
Bond 1	340.0	353.54	[0,1000]
Bond 2	340.0	338.53	[0,1000]
Bond 3	267.8	243.27	[0,1000]
Bond 4	367.0	394.55	[0,1000]
Bond 5	367.0	391.24	[0,1000]
Bond 6	519.7	484.68	[0,1000]
Bond 7	336.8	250.79	[0,1000]
Bond 8	476.7	425.33	[0,1000]
Bond 9	426.7	319.71	[0,1000]
Angle 1	33.0	33.65	[0,100]
Angle 2	33.0	33.78	[0,100]
Angle 3	37.47	44.63	[0,100]
Angle 4	37.47	32.33	[0,100]
Angle 5	69.95	29.58	[0,100]
Angle 6	34.98	47.05	[0,100]
Angle 7	69.95	35.88	[0,100]
Angle 8	69.95	61.86	[0,100]
Angle 9	37.47	46.85	[0,100]
Angle 10	34.98	31.69	[0,100]
Angle 11	34.98	17.64	[0,100]
Angle 12	68.31	99.02	[0,100]
Angle 13	69.95	101.28	[0,100]
Angle 14	69.95	100.05	[0,100]
Dihedral 1	5.97	-2.91	[-30,30]
Dihedral 2	0.16	0.24	[-30,30]

Dihedral 3	7.62	-1.19	[-30,30]
Dihedral 4	3.97	-5.09	[-30,30]
Dihedral 5	0.00	0.03	[-30,30]
Dihedral 6	3.97	3.7	[-30,30]
Dihedral 7	-0.63	1.22	[-30,30]
Dihedral 8	0.06	-0.25	[-30,30]
Dihedral 9	7.62	5.66	[-30,30]
Dihedral 10	-0.85	2.54	[-30,30]
Dihedral 11	5.97	-0.45	[-30,30]
Dihedral 12	5.97	-17.68	[-30,30]
Dihedral 13	3.97	-0.73	[-30,30]
Dihedral 14	3.97	-8.64	[-30,30]
Dihedral 15	0	1.84	[-30,30]
Dihedral 16	7.62	-0.93	[-30,30]
Dihedral 17	7.62	17.56	[-30,30]
Dihedral 18	0.73	1.89	[-30,30]
Dihedral 19	0.095	-1.62	[-30,30]
Fitness	165.85	44.34	

Table 8.8: Original and optimized Im21$^+$ (CLaP) parameters

Besides the large error for the first frequency, the Im21 CLaP parameters seem to optimize reasonably well.

Listing 8.8: Im21 (CLaP) Vibrational frequencies (QM-MM)

```
 31.9379      :   51.2208
 66.1424      :   83.1824
163.9637      :  146.5801
180.7203      :  182.0815
238.8148      :  241.5288
289.4202      :  293.1758
340.5977      :  352.5591
428.1714      :  395.3962
566.6117      :  499.7268
605.2522      :  571.6905
611.5905      :  605.8737
677.3781      :  646.5969
729.0521      :  734.6138
765.3851      :  743.8954
809.6147      :  798.3710
836.2716      :  806.0735
930.9489      :  858.6443
```

```
 991.5424    :  924.3716
1005.5582    :  967.3259
1047.5912    : 1011.1127
1055.3311    : 1014.9376
1075.0166    : 1038.4425
1092.9205    : 1041.7877
1106.1224    : 1089.4511
1121.3484    : 1100.5388
1222.8444    : 1188.0415
1252.4495    : 1211.3720
1272.9654    : 1233.1179
1297.0991    : 1297.5330
1341.2531    : 1339.1239
1351.3738    : 1364.5029
1372.5158    : 1364.8487
1387.3022    : 1372.7058
1398.8874    : 1389.9020
1411.8099    : 1393.4047
1415.9309    : 1449.2416
1425.7842    : 1496.8313
1427.8243    : 1503.4297
1533.9560    : 1547.8920
1537.9800    : 1580.0826
2916.7012    : 2867.3039
2940.5125    : 2931.9456
2945.3400    : 2967.1639
2980.0839    : 2973.9155
3000.1852    : 2975.4814
3003.7189    : 3032.3156
3022.8626    : 3034.3397
3039.6000    : 3035.7556
3140.8446    : 3159.7733
3157.4957    : 3165.5263
3164.6705    : 3179.8794
```

8.9 Summary

Replicating the QM-calculated normal modes with CLaP force fields is more difficult than with CHARMM force fields. In particular, the lowest frequency seems to differ significantly or even vanish. It is possible to use weighted fitness values for the optimization that put emphasis on the lower frequencies, but the overall frequency matching suffers from this. Additionally the ClaP force fields for Im1H$^+$ are very difficult to optimize compared to all other use cases and the final fitness value is not satisfying. However, the improved fitness value is still approximately 50% of the original fitness.

	OAc	HOac	Im1	Im1H	OTf*	OAc*	Im1H*	Im21*
Improvement	87%	81%	67%	69%	60%	67%	52%	73%

Table 8.9: Improvement summary. * = CLaP force field.

8.10 Additional remarks

If the algorithm reaches a point where it improves slowly, it appears to be a good idea to lower the minimum improvement value and step size value in the configuration file. If the first vibrational frequencies are especially important, use weights for the fitness function. This will reduce the overall fitness, but results in better lowest frequencies.

CHAPTER 9

IR spectra

The calculation of an IR spectrum is a computationally expensive task and only done at the end of the optimization process where parameters with a reasonable fitness are available. IR spectra were computed using the original and optimized parameters from chapter 8 for the following molecule/ion pairs:

- Im1/HOAc (CHARMM)
- Im1H$^+$/OAc$^-$ (CHARMM)
- Im1H$^+$/OTf$^-$ (CLaP)
- Im21$^+$/OAc$^-$ (CLaP)
- Im21$^+$/OTf$^-$ (CLaP)

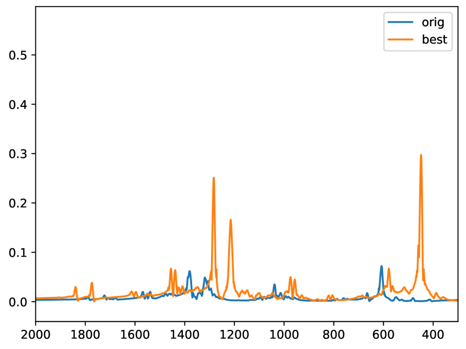
The CLaP force fields for Im1H$^+$/OAc$^-$ ion pairs lead to computation errors and were skipped in this chapter.

9.1 Gas phase

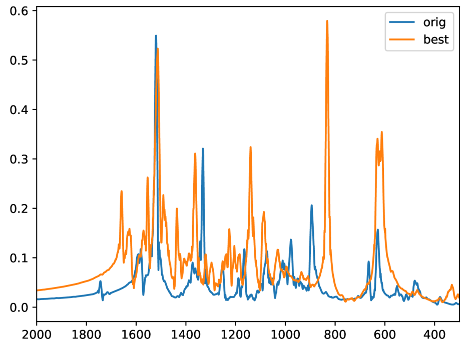
Spectra for *single* molecule pairs were computed in the gas phase and the differences between the spectra resulting from the original parameters and those resulting from the optimized parameters can be seen in the visualizations. The gas phase computations are not intended for evaluation, but rather to show how much the parameter sets differ in the spectrum domain.

9.1.1 Im1/HOAc and Im1H$^+$/OAc$^-$

Figure 9.1 (a) shows three peaks (around 450, 1200, and 1300 cm^{-1}) with a large magnitude that appear with the new set of parameters for Im1/HOAc while figure 9.1 (b) shows that three significant peaks (around 600, 1300, and 1500 cm^{-1}) are retained. The IR spectra based on the original and optimized parameters appear to differ more for the uncharged molecules than for the charged molecules.



(a) IR spectrum of Im1/HOAc (CHARMM).



(b) IR spectrum of Im1H$^+$/OAc$^-$ (CHARMM).

Figure 9.1: IR spectra of Im1/HOAc and Im1H$^+$/OAc$^-$ as single molecule pairs in gas phase with original (blue) and improved (orange) parameters. The x-axis represents the wavenumber in cm^{-1} and the y-axis the intensity.

9.1.2 Im1H$^+$/OTf$^-$ (CLaP)

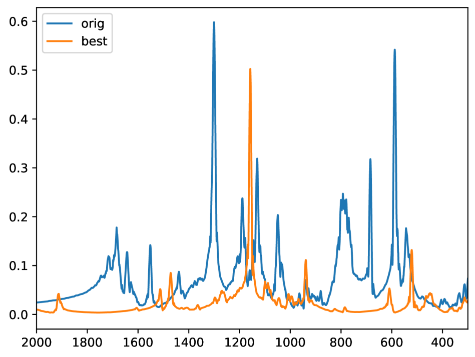


Figure 9.2: IR spectrum of Im1H$^+$/OTf$^-$ (CLaP) with old parameters (blue) and new parameters (orange).

Figure 9.2 shows that high intensity peaks from the original parameter set disappear with the optimized parameters. There is little similarity between both spectra, probably due to the fact that the Im1H CLaP parameters were particularly difficult to optimize.

9.1.3 Im21$^+$/OTf$^-$ (CLaP)

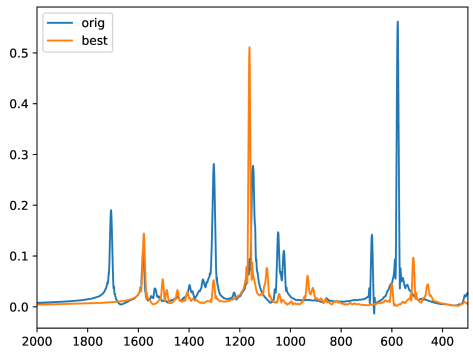


Figure 9.3: IR spectrum of Im21$^+$/OTf$^-$ (CLaP) with old parameters (blue) and new parameters (orange).

Figure 9.3 shows that the peaks below 1200 cm^{-1} and at 1600 cm^{-1} are conserved. The frequencies at 600, 1050, and 1300 cm^{-1} have a lower intensity with the new parameters. Another interesting observation is that the most left peak from the original dataset disappears completely.

9.1.4 Im21$^+$/OAc$^-$ (CLaP)

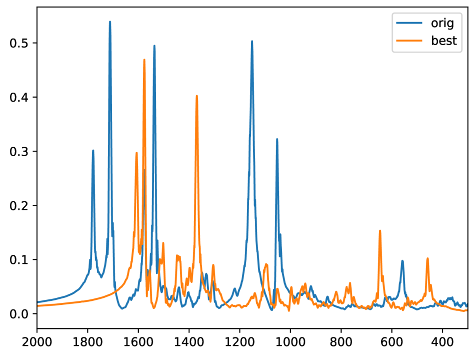


Figure 9.4: IR spectrum of Im21$^+$/OAc$^-$ (CLaP) with old parameters (blue) and new parameters (orange).

In figure 9.4 we see almost an exact overlap at 1600 cm^{-1} like in the previous example. However, in general there seems to be a shift of 200 cm^{-1} towards lower wavenumbers for the twin peaks on the left.

9.2 Liquid phase

500 molecule pairs were computed in the liquid phase and their spectra were compared to the QM-calculated frequencies. The spectra are shown for each of the molecules as well as the combined spectrum.

9.2.1 Im1/HOAc

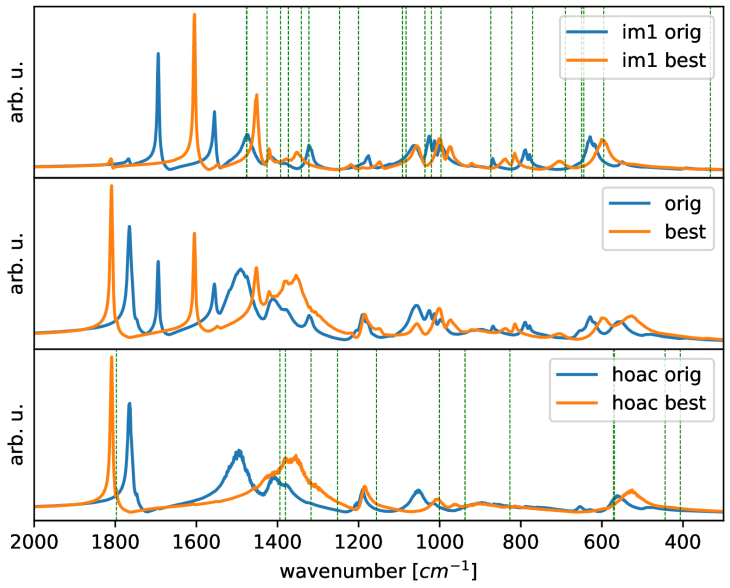


Figure 9.5: Simulated liquid phase spectra of Im1/HOAc. Improved and old spectrum are shown for Im1 (top), HOAc (bottom), and combined (middle). Green dashed lines represent QM-calculated vibrational frequencies for separate molecule types.

In figure 9.5 the original (blue) and improved (orange) spectra of Im1 and HOAc are shown separately for each molecule, and the QM-calculated frequencies are plotted in green. The combined spectrum is shown in the middle. For Im1 some peaks are more accurate with the new parameters, for example the top right peak, while others deviate slightly more. For HOAc significant improvements can be seen. The new spectrum matches the QM-calculated vibrational frequency on the left much more accurately while some false peaks on the old spectrum are not found in the new one, for example at 1500 or at 1050 cm^{-1}.

9.2.2 Im1H$^+$/OAc$^-$

Figure 9.6 shows good results for Im1H$^+$ and OAc$^-$ as both have peaks that align better with the QM-calculated frequencies shown by the green dashed lines. This is especially

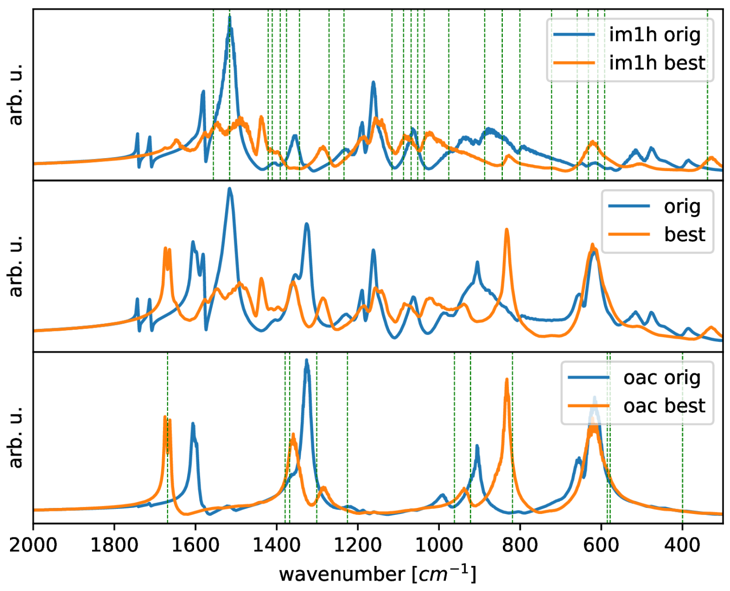


Figure 9.6: Simulated liquid phase spectra of Im1H$^+$/OAc$^-$.

obvious for OAc$^-$ at wavenumbers around 800, 1400, and 1600. The combined spectra show significant differences between parameters, with the exception of the area above 600 cm$^-$1, which appears to align well.

9.2.3 Im1H$^+$/OTf$^-$ (CLaP)

Figure 9.7 shows that the Im1H$^+$ spectrum resulting from the new Im1H$^+$ CLaP parameters aligns surprisingly well on the peaks around 1500, 1300, 1000 cm^{-1}, and the smaller peaks around 1400 and around 600 to 900 cm^{-1}. The three large spikes on the left from the original parameters disappear. The OTf$^-$ spectrum matches even better, with the peaks slightly over 900 cm^{-1} and slightly below 1200 cm^{-1} aligning almost exactly with the QM-calculated frequencies, and those between 400 and 600 cm^{-1} still aligning decently. Overall, the combined spectrum looks different for the old and the new parameters.

9.2.4 Im21$^+$/OAc$^-$ (CLaP)

In Figure 9.8 we see that one improvement for Im21$^+$ is the absence of spectral lines from the original parameter sets at 1700 and 1100 cm$^-$1. It is more difficult to evaluate the remaining QM spectral lines, but they seen to roughly align with the spectrum. The evaluation is simpler for OAc$^-$ and the improved spectrum aligns better with the spectral lines than previously.

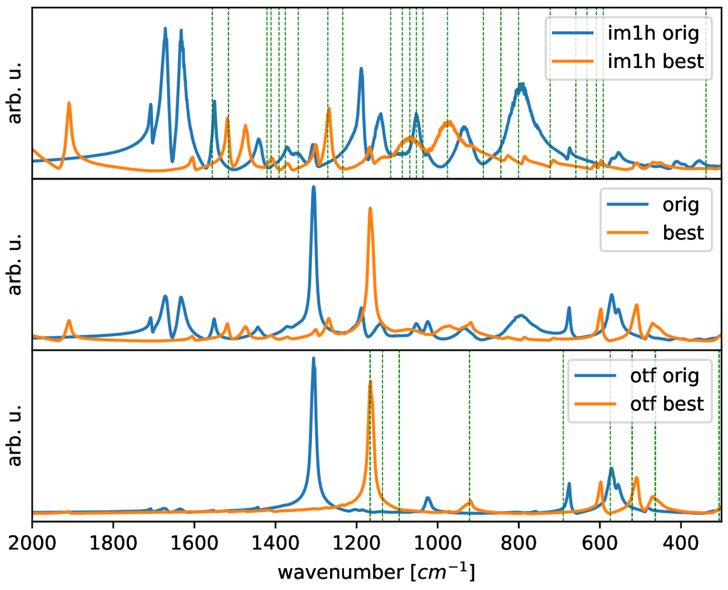


Figure 9.7: Simulated liquid phase spectra of Im1H$^+$/OTf$^-$ (CLaP).

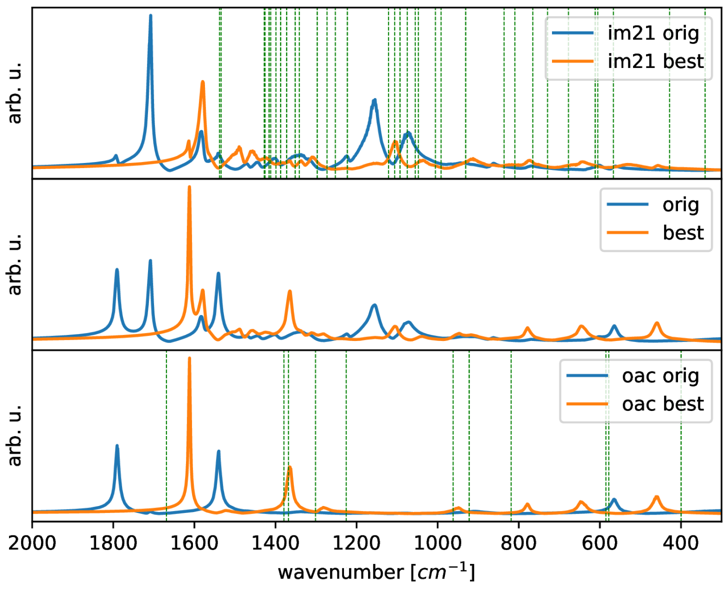


Figure 9.8: Simulated liquid phase spectra of Im21$^+$/OAc$^-$ (CLaP).

9.2.5 Im21$^+$/OTf$^-$ (CLaP)

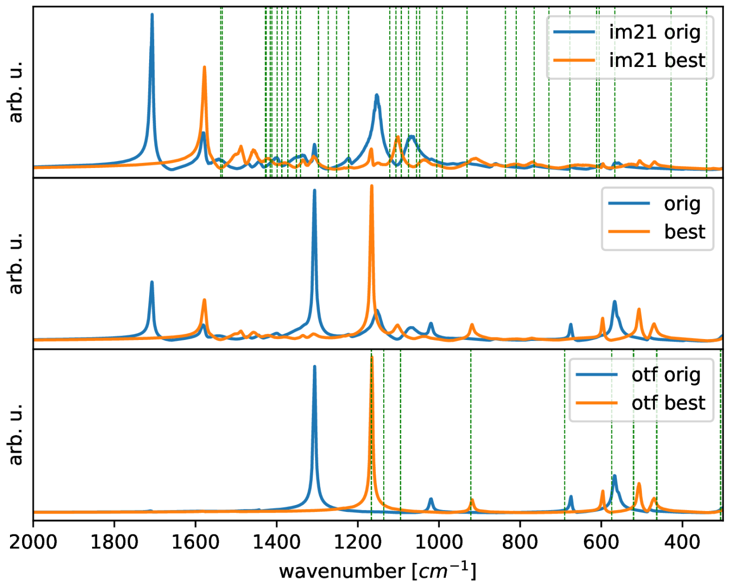


Figure 9.9: Simulated liquid phase spectra of Im21$^+$/OTf$^-$ (CLaP).

The interpretation is the same as for the previous two examples for Im21$^+$ and OTf$^-$. The combined spectrum shows a shift to the right for many significant peaks, but they are otherwise very similar.

9.3 Comparison to physical measurements

In this section the simulated spectra are compared to the physically measured IR spectrum of the ionic liquid Im1H$^+$/OAc$^-$. Spectra were computed for both, pure liquids, and for 30:70 mixtures of Im1/HOAc (which are neutral) and Im1H$^+$/OAc$^-$ (which are ionic). The reason why 30% and 70% mixtures are chosen is because the equilibrium produced by proton transfers is assumed to be at this point.

In figure 9.10 we can see that the original parameters produce a 30:70 spectrum closer to the experiment than the improved parameters, at least for high wavenumbers. It is difficult to recognize a significant improvement compared to the original spectra.

Surprisingly, the improved spectrum for the pure ionic liquid has a better alignment with the experimental spectrum than the 30:70 mixture, see figure 9.11. Note the (double) peak close to the experimental peak at 1700, below 1400, above 1400, around 1300 and 1100 cm^{-1}, and below (visible in the *ionic best* spectrum in orange in the middle chart).

However, it would also be possible that the peaks in the 30:70 mixtures are actually shifted by 100 cm^{-1} which would again lead to a decent alignment.

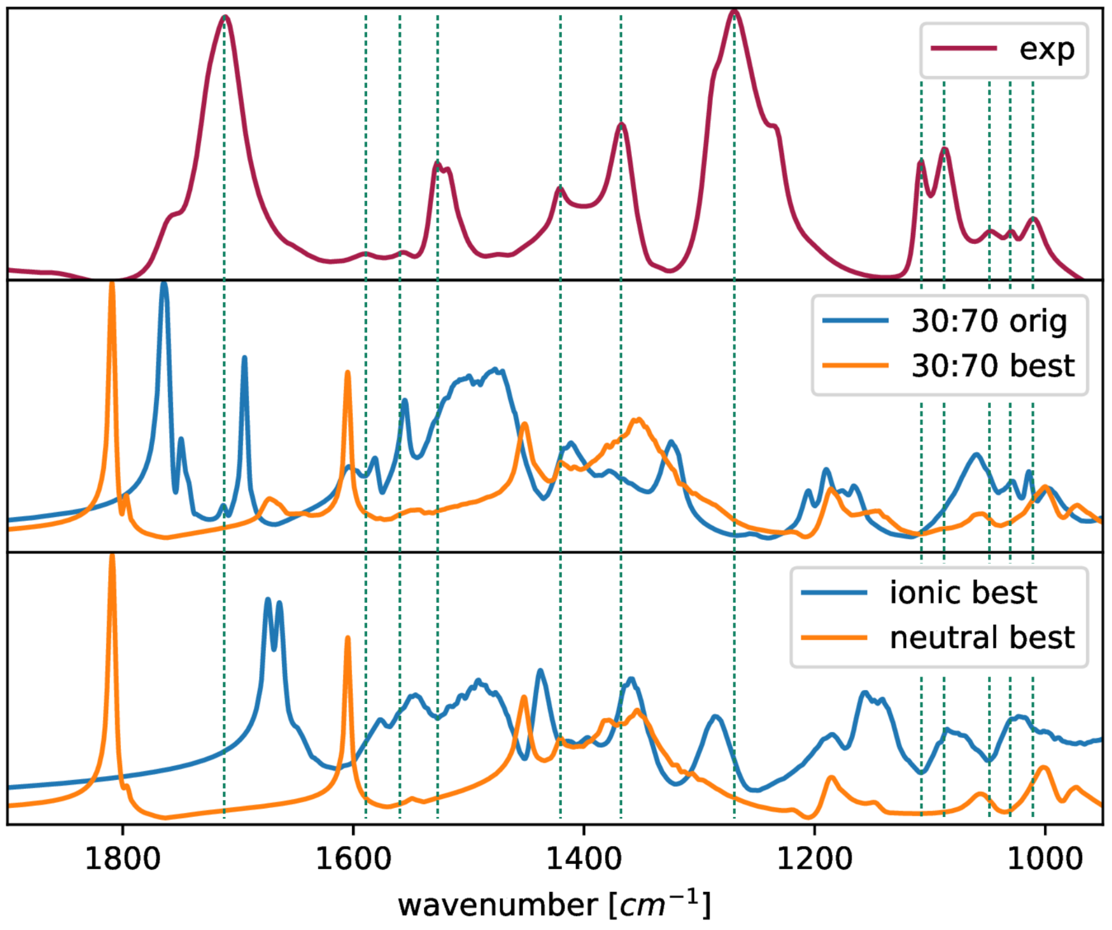


Figure 9.10: The top chart represents the experimental spectrum. The middle chart shows the original and improved simulated spectra for the 30:70 mixtures. The bottom chart shows the spectra resulting from the new (best) parameters for the ionic and neutral liquids respectively. All green dashed lines cross through the peaks of the experimental spectrum in order to allow easier comparison.

The comparison of the computational and experimental IR spectra in liquid phase demonstrates that a parametrization on the pure basis of QM calculations in the gas phase may not be sufficient to predict intramolecular vibrations in the liquid phase. Consequently, a parametrization based on the agreement between computational and experimental spectra in the liquid phase would be superior. However, this is out of reach for the genetic algorithm in the present work as the generation of the MD based IR spectrum takes too much time and hence cannot be performed for each iteration step.

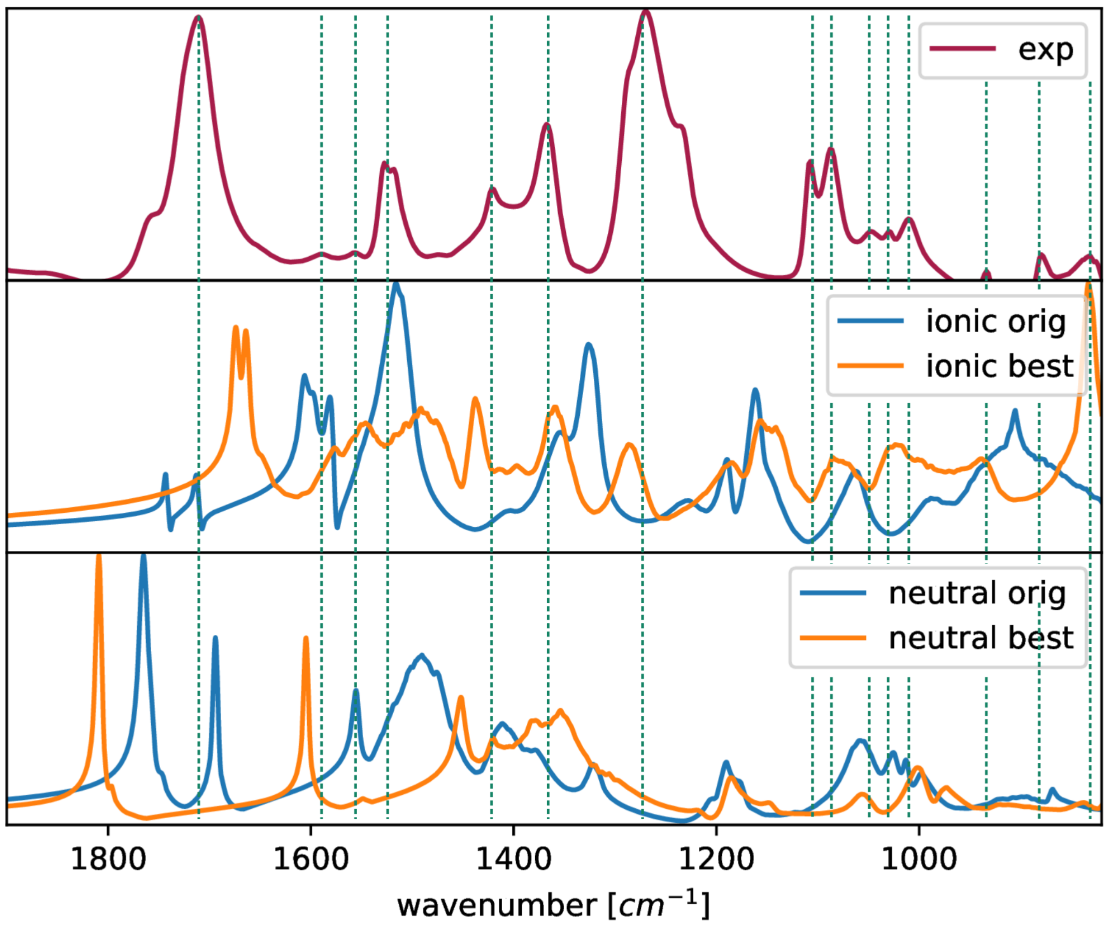


Figure 9.11: The top chart shows again the experimentally measured spectrum of the ionic liquid. The middle chart shows a comparison between the original spectrum and improved spectrum for the pure ionic liquid (Im1H$^+$/OAc$^-$). The bottom chart shows the same for the pure neutral liquid (Im1/HOAc).

CHAPTER 10

Experiments

In this chapter the performance of different FF_GenOpt configurations is compared. The change of the best fitness over time is plotted. However, the unit of time is not seconds but the number of fitness function evaluations, because the real time can vary based on system load. Note that genetic algorithms can have vastly different convergence behavior in different runs even with the same settings as shown in the Paramfit paper [12].

10.1 Part 1

The first part of the test consists of setting a baseline parameter and iteratively changing parts of it. These parameter settings are applied to all systems and the result is plotted on a linear-log scale.

Run #	1	2	3	4	5	6	7
Population size	30	**200**	30	30	30	30	**200**
Mutations/Generation	10	**30**	10	10	10	10	**50**
Step size	0.05	0.05	**0.001**	0.05	0.05	0.05	0.05
Crossover BLX	0.5	0.5	0.5	**1.0**	0	0	**0.7**
Crossover SBX	0.5	0.5	0.5	0	**1.0**	0	**0.25**
Crossover Uniform	0	0	0	0	0	**1.0**	**0.05**
Minimum improvement	0.001	0.001	0.001	0.001	0.001	0.001	**0.0001**
Minimum diversity	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001

Table 10.1: Different parameters for the experimental algorithm runs.

OAc$^-$ In figure 10.1 we can see that the algorithm converges in different minima for different runs. Run 2 and Run 7, which are both configurations with a population size of 200, deliver the best results within 25000 evaluations. Run 3, with a small step size, improves slower and suffers from getting stuck in local minima. Run 5 and 6, where SBX and uniform crossovers were used, perform worse than Run 4, where only BLX is used, or Run 1 where a combination of crossover operators is used.

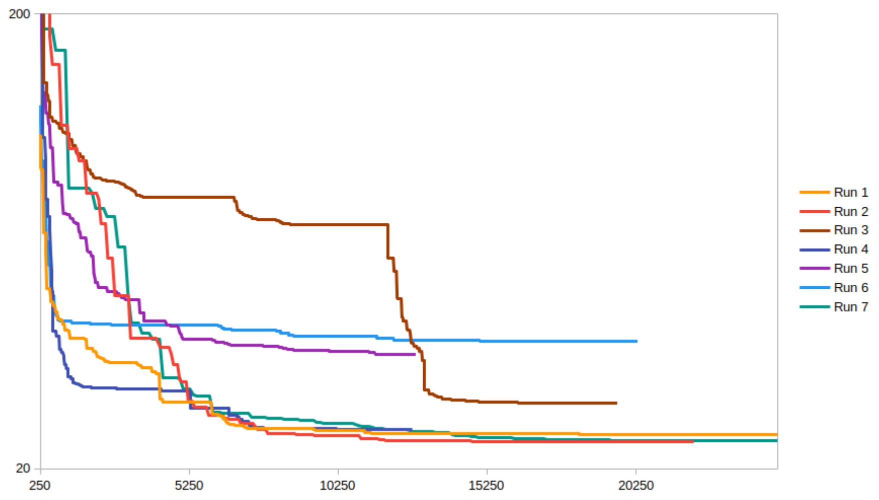


Figure 10.1: Optimizing OAc$^-$ force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

HOAc In figure 10.2 we can again see that Run 2 and Run 7 perform the best. Uniform crossover (Run 6) leads to premature convergence.

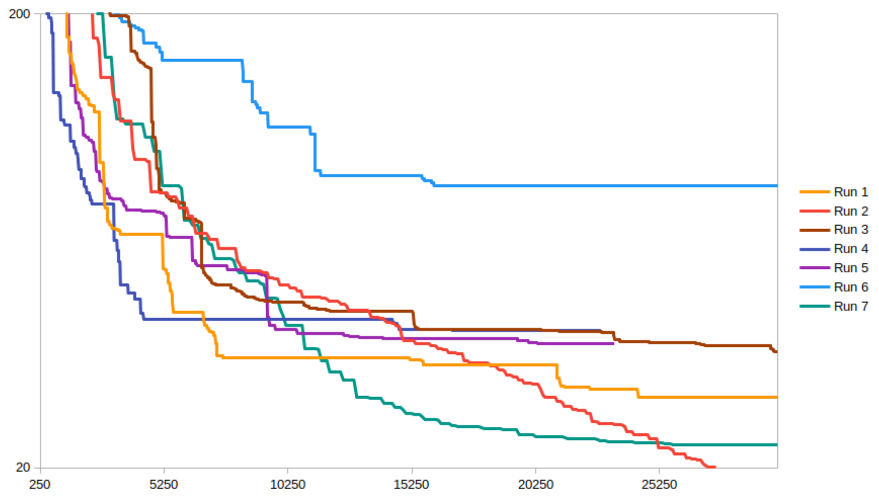


Figure 10.2: Optimizing HOAc force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

Im1 The force field requires significantly more parameters than the previous molecules and therefore requires more function evaluations. Figure 10.3 shows that Run 7 performs the best within 25000 evaluations while Run 2 gets stuck in premature convergence. The small step size in Run 3 also leads to early stagnation.

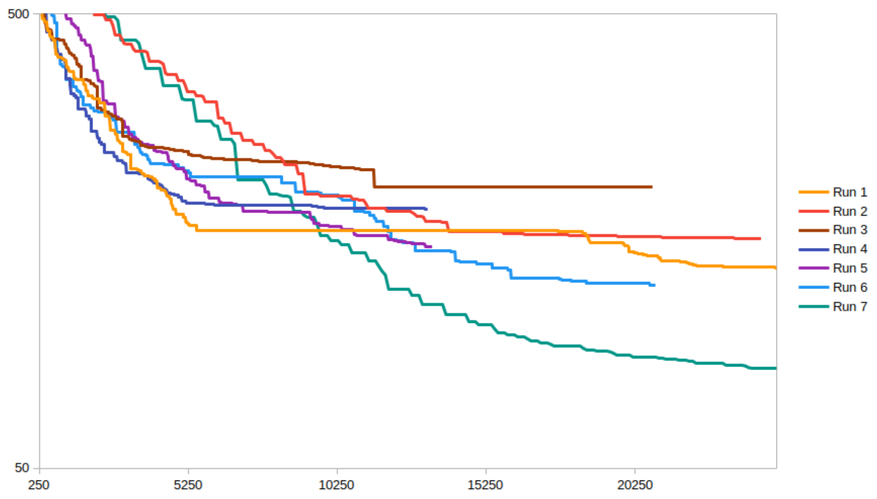


Figure 10.3: Optimizing Im1 force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

Im1H$^+$ The results from figure 10.4 are very similar to Im1 and indicate that a larger population size is beneficial.

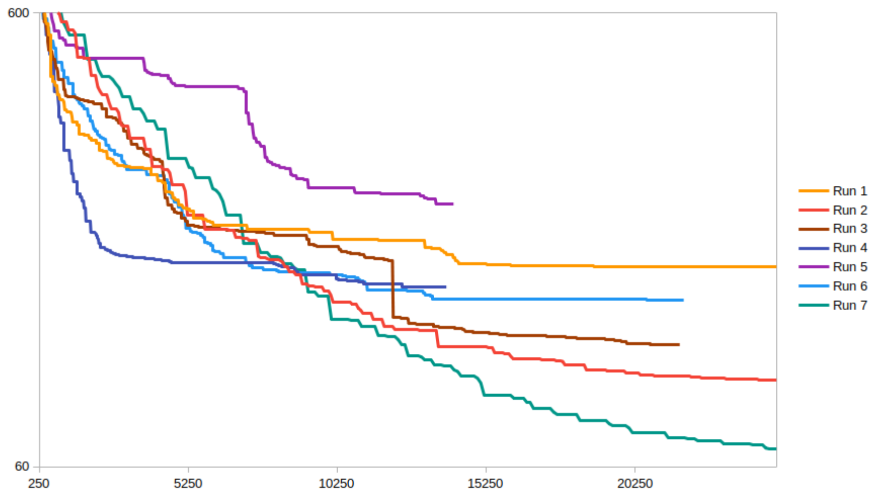


Figure 10.4: Optimizing Im1H$^+$ force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

OTf$^-$ (CLaP) Surprisingly, the best performance in figure 10.5 is achieved by only using uniform crossovers, which otherwise perform poorly. The second best result is calculated in Run 7.

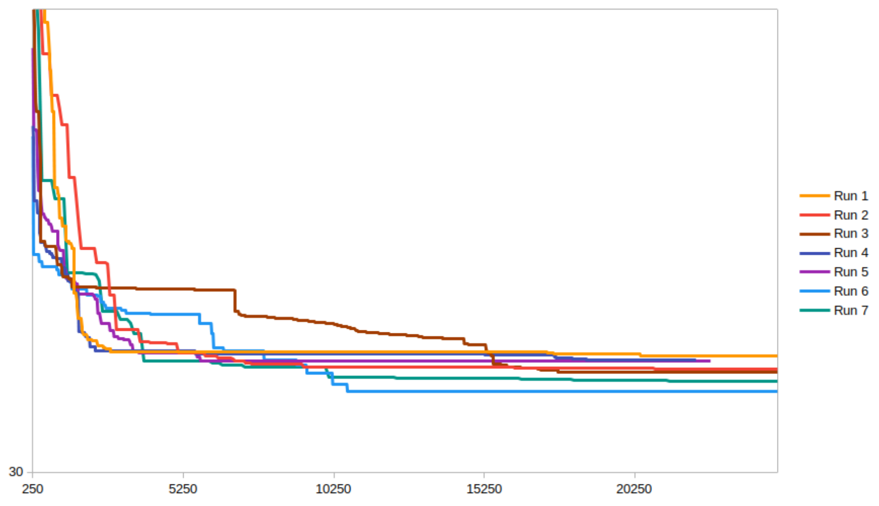


Figure 10.5: Optimizing OTf$^-$ (CLaP) force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

OAc$^-$ (CLaP) Run 1 (orange) in figure 10.6 shows that convergence can be deceptive in genetic algorithms and fitness can improve after a long period of stagnation. However, the best fitness is again reached by Run 2 and Run 7.

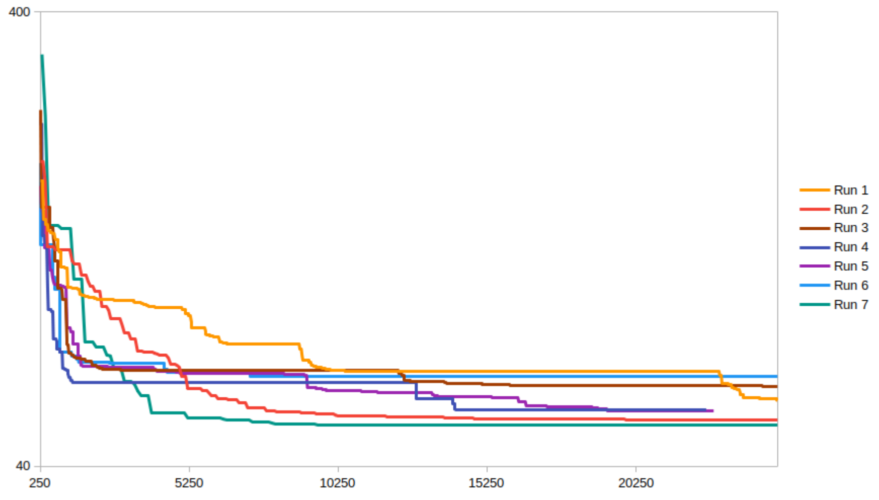


Figure 10.6: Optimizing OAc$^-$ (CLaP) force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

Im1H$^+$ (CLaP) Figure 10.7 shows again that improvement can continue after many fruitless evaluation steps. Run 2 and Run 7 reach the best fitness again.

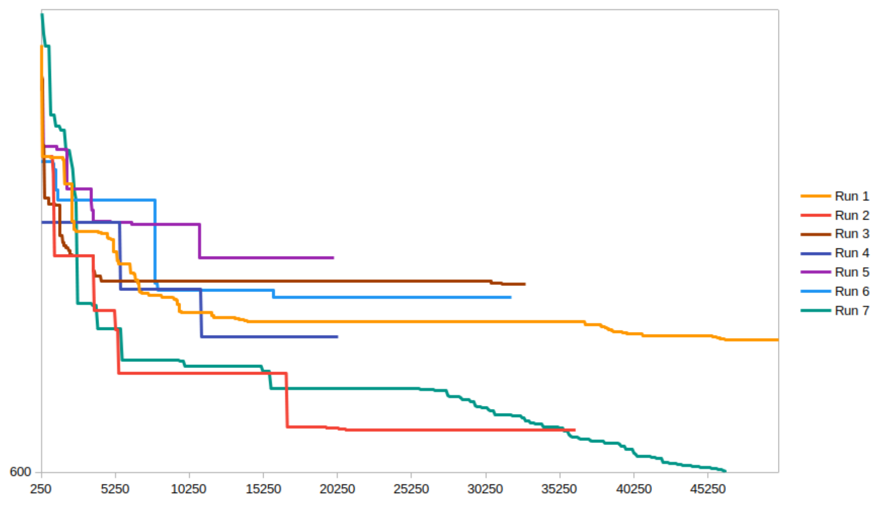


Figure 10.7: Optimizing Im1H$^+$ (CLaP) force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

Im21$^+$ (CLaP) The curves in figure 10.8 are similar to previous ones.

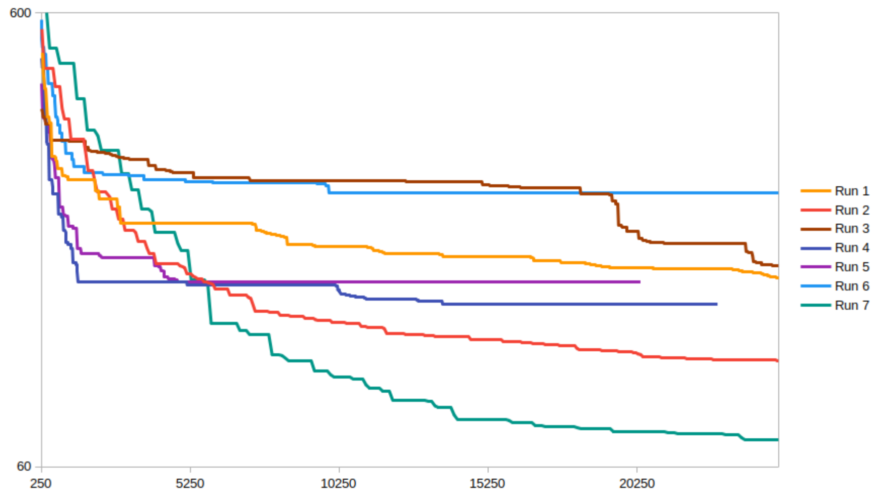


Figure 10.8: Optimizing Im21 (CLaP) force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

This experiment leads to the following observations:

1. A larger population size of 200 works better than a smaller population size of 30.

2. A maximum step size of 0.0001 leads to worse performance than a step size of 0.05 due to getting trapped in small local minima.
3. BLX crossovers tend to work well and uniform crossover tend to behave poorly (except for OTf$^-$). Using a mixture of crossover operations seems reasonable.

10.2 Part 2

The first part establishes that a population size of 200 works better in most cases than a population size of 30. The idea behind this section is to find out how many mutations per generation should be computed. The new settings in table 10.2 are based on Run 7, which achieved the best performance in the previous experiments.

Run #	8	9	10
Population size	200	200	200
Mutations/Generation	**10**	**100**	**200**
Step size	0.05	0.05	0.05
Crossover BLX	0.7	0.7	0.7
Crossover SBX	0.25	0.25	0.25
Crossover Uniform	0.05	0.05	0.05
Minimum improvement	0.0001	0.0001	0.0001
Minimum diversity	0.0001	0.0001	0.0001

Table 10.2: Running the algorithm with different numbers of mutations per generation.

OAc$^-$ In the trial runs the algorithm achieves the best fitness with a mutation value of 50 or 100 for OAc$^-$, see figure 10.9.

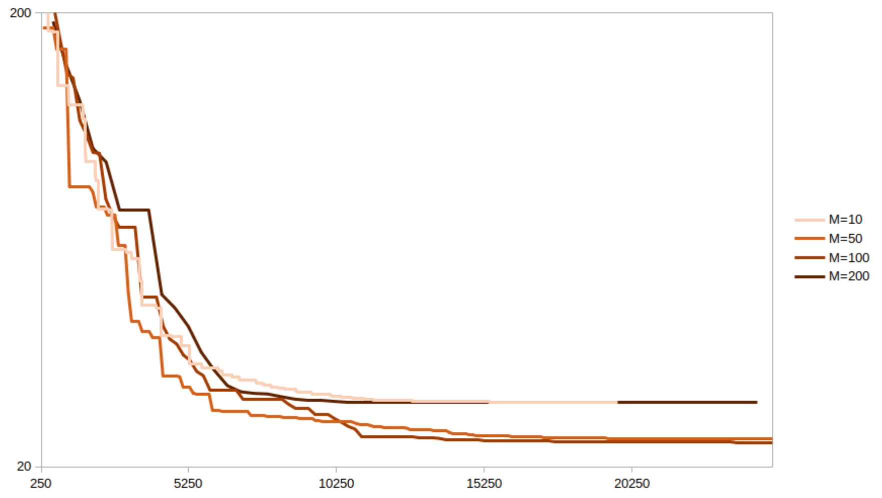


Figure 10.9: Optimizing OAc$^-$ force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

HOAc Figure 10.10 shows that in the case of HOAc a mutation parameter of 100 or 200 is preferable.

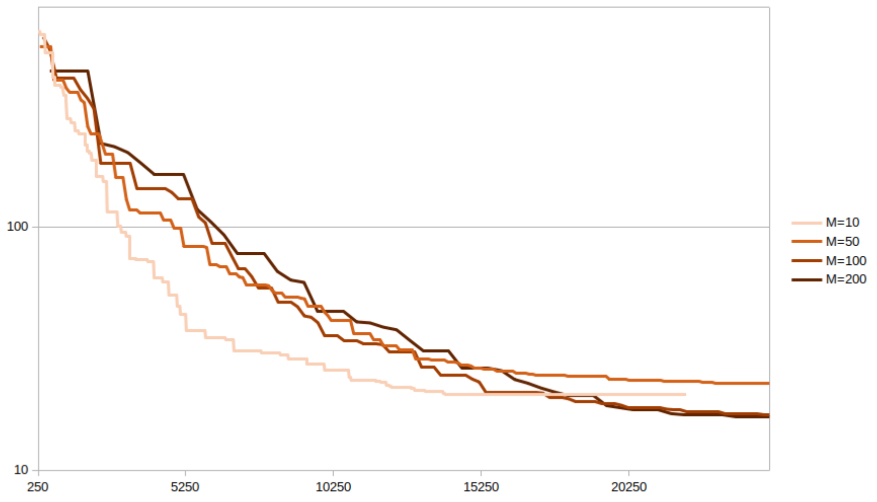


Figure 10.10: Optimizing HOAc force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

Im1 A mutation parameter of M = 50 achieves the best results while M = 10 improves very slowly, see figure 10.11.

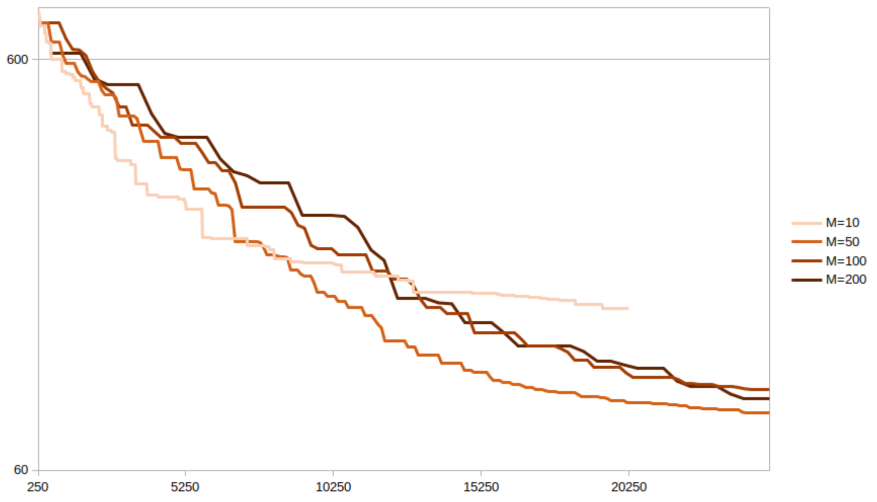


Figure 10.11: Optimizing Im1 force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

Im1H$^+$ The curves in figure 10.12 look remarkably similar to the previous ones (figure 10.11 and lead to the same conclusion.

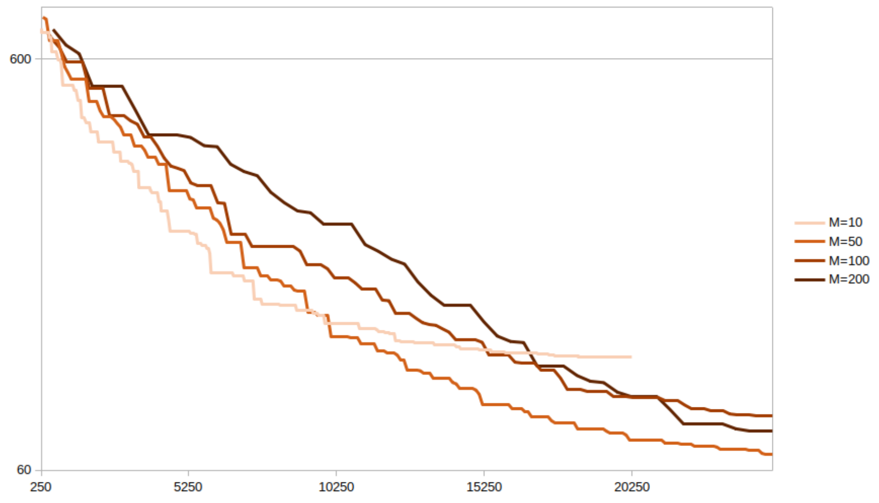


Figure 10.12: Optimizing Im1H$^+$ force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

OTf⁻ (CLaP) Mutation settings of M = 50 also deliver the best performance for Triflate force field as shown in figure 10.13.

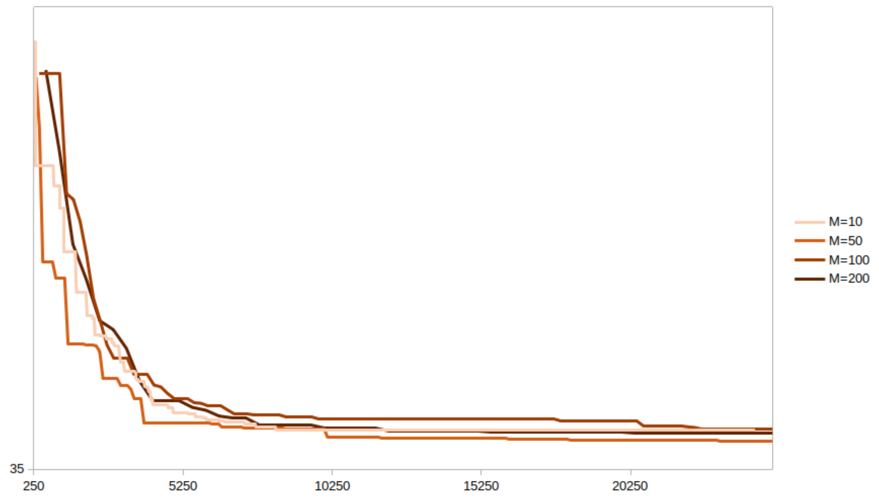


Figure 10.13: Optimizing OTf⁻ (CLaP) force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

OAc⁻ (CLaP) It is difficult to see a clear winner in figure 10.14 since all runs quickly converge to a fitness value of 49.05.

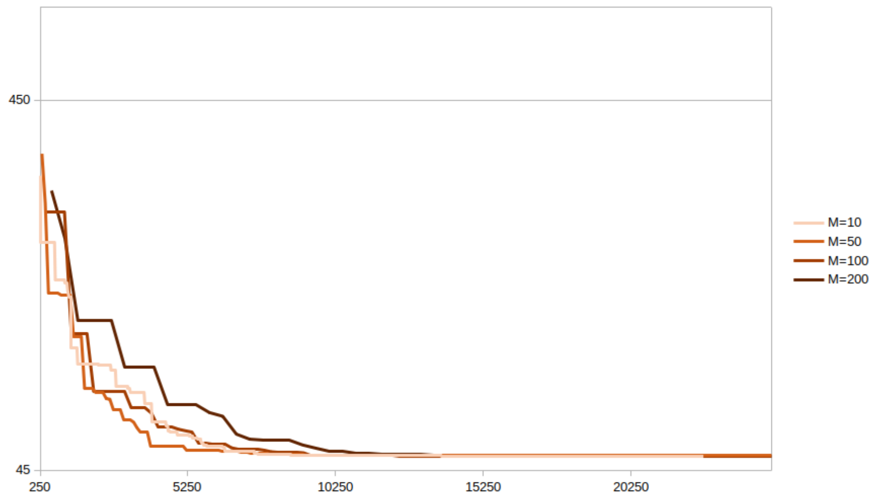


Figure 10.14: Optimizing OAc- (CLaP) force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

Im1H$^+$ (CLaP) Finding the right CLaP force field parameters for Im1H$^+$ seems more difficult than for the other molecules. M = 10 seems to improve the fitness (see figure 10.15), but the validity of the resulting force field parameters is questionable.

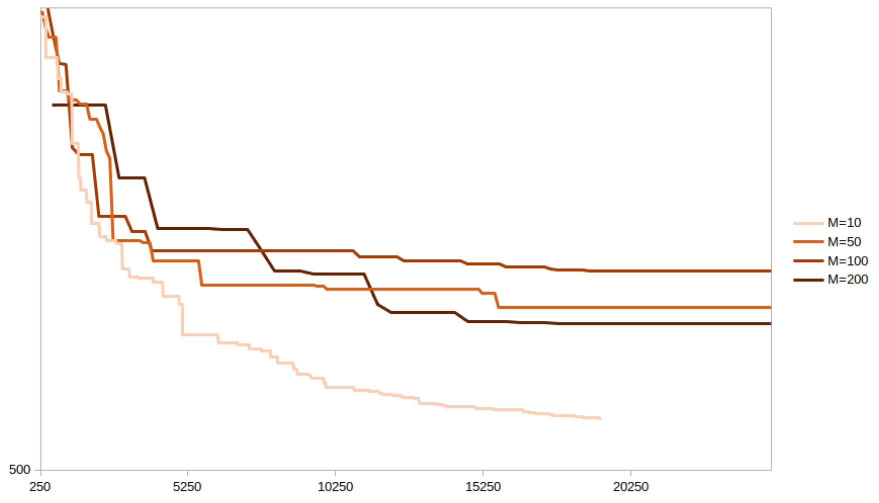


Figure 10.15: Optimizing Im1H- (CLaP) force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

Im21$^+$ (CLaP) While figure 10.16 shows that M = 50 results in the best fitness (after 25000 function evaluations), the results are very close. Only M = 10 performs poorly by stagnating early.

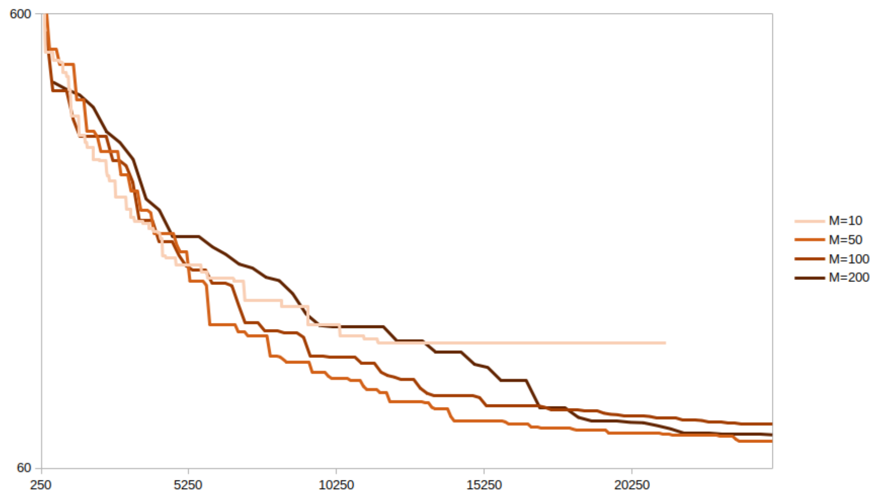


Figure 10.16: Optimizing Im21 (CLaP) force fields with different parameters. The y-axis is the fitness and the x-axis is the number of function evaluations.

Based on these observations two more conclusions can be made:

1. Setting the number of mutations per generation parameter to 50 seems to work well for all cases except for HOAc and $Im1H^+$ (CLaP).
2. Premature convergence can occur even with larger populations, which is especially noticeable at OAc^- (CLaP). That does not mean that the algorithm will never improve again, but long periods of stagnation might precede further improvements.

10.3 Summary

Hyperparameter analysis with genetic algorithms is a difficult task due to the inherent randomness and computational effort that is required each test run. Additionally, sudden fitness jumps can occur after long periods of stagnation and the same parameters can lead to different results and convergence speeds. For the examples in this chapter the parameters from Run 7 in table 10.1 seem like a reasonable starting point.

CHAPTER 11

Conclusion

The success of machine learning algorithms based on deep neural networks has pulled attention away from genetic algorithms. This project shows that genetic algorithms, in combination with other optimization methods, are capable of optimizing highly complex optimization surfaces. Fitness values for all force field parameters have been significantly improved. However, the IR spectra still need improvement and there is potential for developing more sophisticated approaches.

In future work, the fitness function could be adapted in such a way that its value predicts the results of IR spectral analysis more accurately. This could be done, for example by, using machine learning and creating a substitute function. More sophisticated algorithms than the random search algorithm could also improve the convergence speed. Parallelization is another option to enhance performance. Faster NMA and fitness evaluations could be implemented. Functional forms of the force fields with a better reproduction of IR spectra could be used.

The experiments have shown that the genetic algorithm converges to different search space areas. Advanced genetic algorithms can manage multiple populations while maintaining inter-population diversity in order to find multiple local (or global) minima. Alternatively, a particle swarm approach could be used.

List of Figures

List of Tables

Bibliography

[1] Peter Atkins and Julio Paula. *Atkins' physical chemistry.* Oxford University press, 2008.

[2] A.R. Leach. *Molecular Modelling: Principles and Applications.* Prentice Hall, 2001.

[3] David G Lovering and Douglas Inman. *Ionic Liquids.* Plenum Publishing Corporation, 1981.

[4] Sandip K. Singh and Anthony W. Savoy. Ionic liquids synthesis and applications: An overview. *Journal of Molecular Liquids*, 297:112038, 2020.

[5] B. R. Brooks, C. L. Brooks, A. D. Mackerell, L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A. R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R. W. Pastor, C. B. Post, J. Z. Pu, M. Schaefer, B. Tidor, R. M. Venable, H. L. Woodcock, X. Wu, W. Yang, D. M. York, and M. Karplus. Charmm: The biomolecular simulation program. *Journal of Computational Chemistry*, 30(10):1545–1614, 2009.

[6] National Institute of Standards and Technology. The NIST Reference on Constants, Units, and Uncertainty. `https://physics.nist.gov/cgi-bin/cuu/Value?mpsme`, 2019. Online; accessed 22-July-2021.

[7] Frank Jensen. *Introduction to Computational Chemistry.* John Wiley and Sons, Inc., Hoboken, NJ, USA, 2006.

[8] Christian Schröder and Othmar Steinhauser. Simulating polarizable molecular ionic liquids with drude oscillators. *The Journal of Chemical Physics*, 133(15):154511, 2010.

[9] S. Bienz, L. Bigler, T. Fox, and H. Meier. *Spektroskopische Methoden in der organischen Chemie.* Thieme, 2016.

[10] Tamar L Greaves and Calum J Drummond. Protic ionic liquids: properties and applications. *Chemical reviews*, 108(1):206–237, 2008.

[11] Andrea C. Vaiana, Zoe Cournia, Ion Bogdan Costescu, and Jeremy C. Smith. AFMM: A molecular mechanics force field vibrational parametrization program. *Comput. Phys. Commun.*, 167(1):34–42, 2005.

[12] Robin M. Betz and Ross C. Walker. Paramfit: Automated optimization of force field parameters for molecular dynamics simulations. *J. Comput. Chem.*, 36(2):79–87, 2015.

[13] Romelia Salomon-Ferrer, David A Case, and Ross C Walker. An overview of the amber biomolecular simulation package. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 3(2):198–210, 2013.

[14] Leonard Andreevič Rastrigin. Systems of extremal control. *Nauka*, 1974.

[15] Ingo Rechenberg. Evolutionsstrategie. problemata, 1973.

[16] John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, 1975. second edition, 1992.

[17] Michael Affenzeller, Stephan Winkler, Stefan Wagner, and Andreas Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications.* Chapman and Hall/CRC, 1st edition, 2009.

[18] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[19] Alberto Martin, Richard Chahwan, Jahan Yar Parsa, and Matthew D. Scharff. Chapter 20 - somatic hypermutation: The molecular mechanisms underlying the production of effective high-affinity antibodies. In Frederick W. Alt, Tasuku Honjo, Andreas Radbruch, and Michael Reth, editors, *Molecular Biology of B Cells (Second Edition)*, pages 363–388. Academic Press, London, second edition edition, 2015.

[20] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989.

[21] E. Wirsansky. *Hands-On Genetic Algorithms with Python: Applying genetic algorithms to solve real-world deep learning and artifical intelligence problems.* Packt Publishing, 2020.

[22] Vijay Chahar, Sourabh Katoch, and Sumit Chauhan. A review on genetic algorithm: Past, present, and future. *Multimedia Tools and Applications*, 80, 2021.

[23] Tadahiko Murata, Hisao Ishibuchi, and Hideo Tanaka. Multi-objective genetic algorithm and its applications to flowshop scheduling. *Computers and Industrial Engineering*, 30(4):957–968, 1996.

[24] L. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. In *FOGA*, 1992.

[25] Kalyanmoy Deb and Ram Bhushan Agrawal. Simulated binary crossover for continuous search space. *Complex Syst.*, 9(2), 1995.

[26] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.*, 1(1):67–82, 1997.

[27] José N Canongia Lopes, Johnny Deschamps, and Agílio AH Pádua. Modeling ionic liquids using a systematic all-atom force field. *The Journal of Physical Chemistry B*, 108(6):2038–2047, 2004.

[28] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[29] Thomas Torsney-Weir, Michael Sedlmair, and Torsten Möller. Sliceplorer: 1D slices for multi-dimensional continuous functions. *Comput. Graph. Forum*, 36(3):167–177, 2017.

[30] David Mautner Himmelblau. *Applied Nonlinear Programming.* McGraw-Hill, 1972.

[31] Mario Köppen. The curse of dimensionality. In *5th Online World Conference on Soft Computing in Industrial Applications (WSC5)*, volume 1, pages 4–8, 2000.

Appendix A

This part contains axis-parallel slices of all optimization surfaces discussed in the thesis.

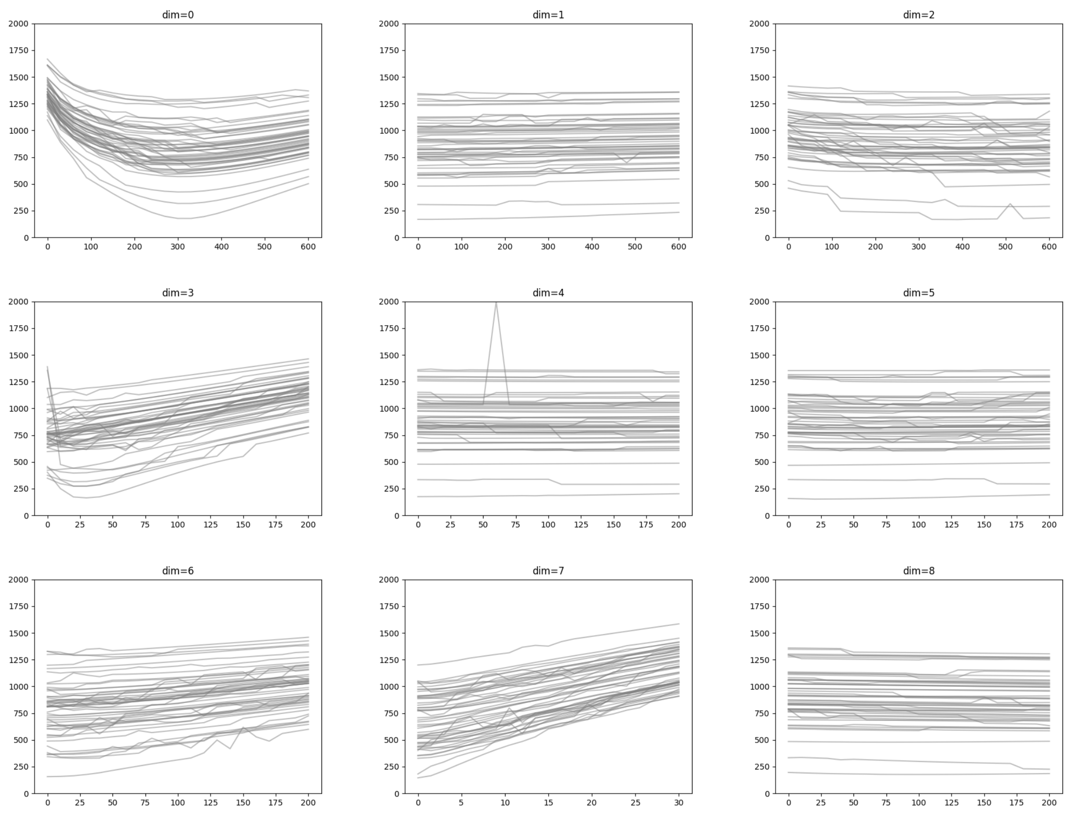


Figure 1: OAc

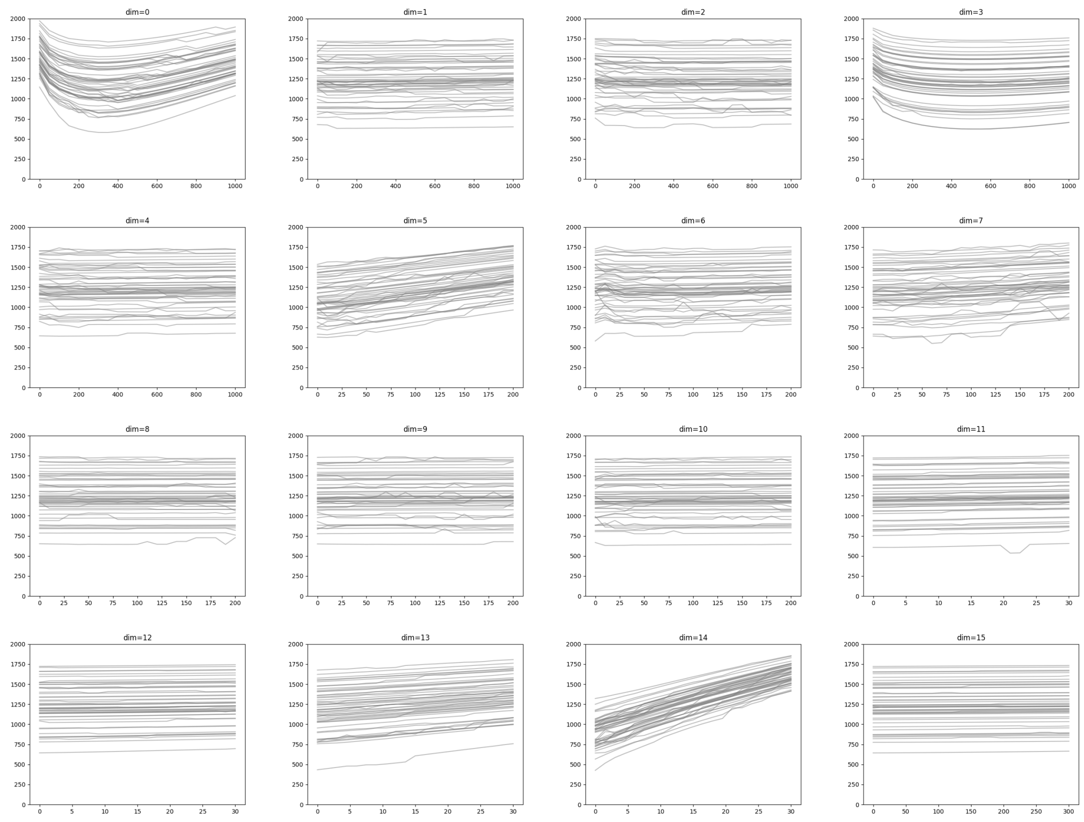


Figure 2: HOAc

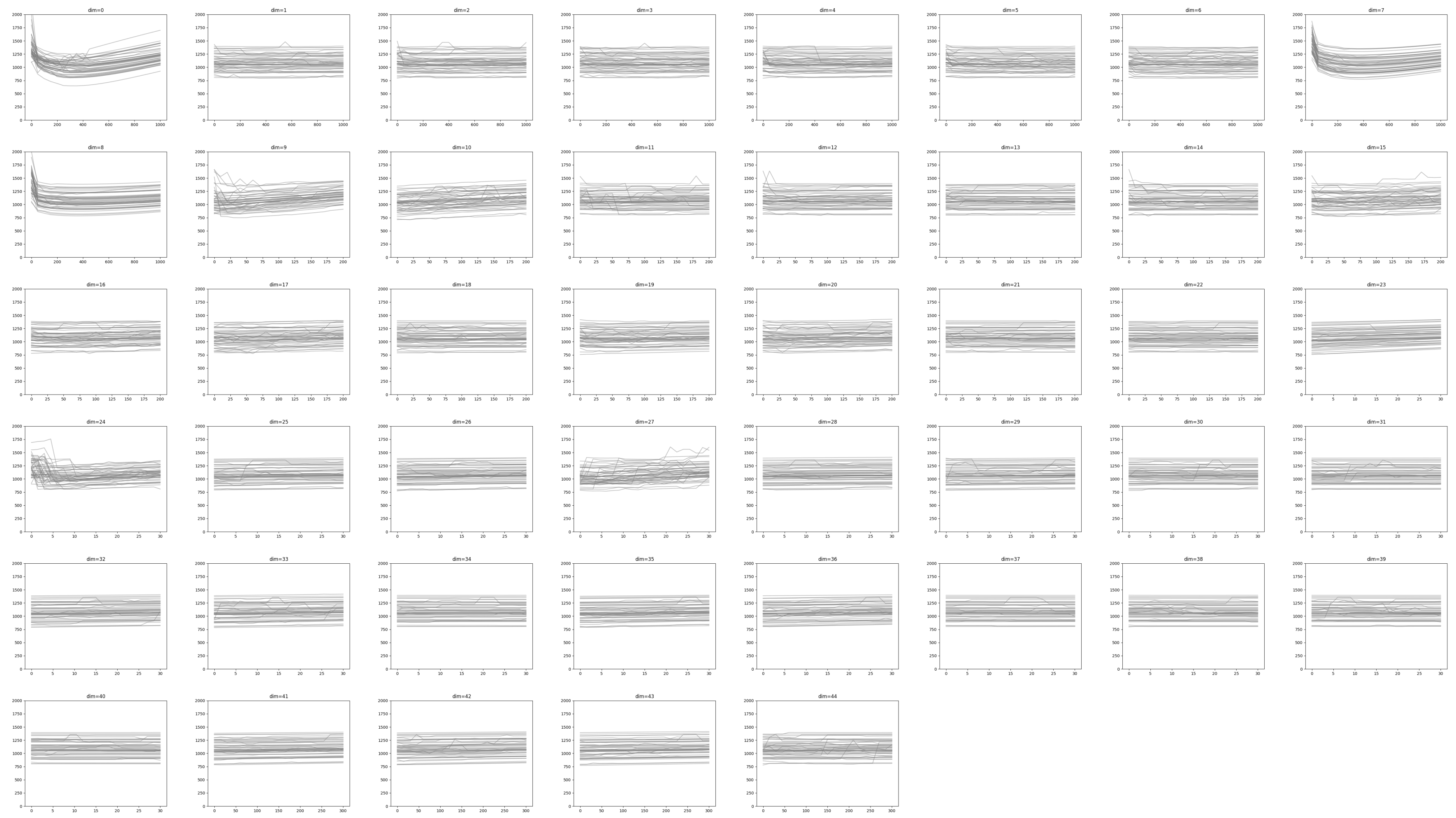

dim=0
dim=1
dim=2
dim=3
dim=4
dim=5
dim=6
dim=7
dim=8
dim=9
dim=10
dim=11
dim=12
dim=13
dim=14
dim=15
dim=16
dim=17
dim=18
dim=19
dim=20
dim=21
dim=22
dim=23
dim=24
dim=25
dim=26
dim=27
dim=28
dim=29
dim=30
dim=31
dim=32
dim=33
dim=34
dim=35
dim=36
dim=37
dim=38
dim=39
dim=40
dim=41
dim=42
dim=43
dim=44

Figure 3: Im1

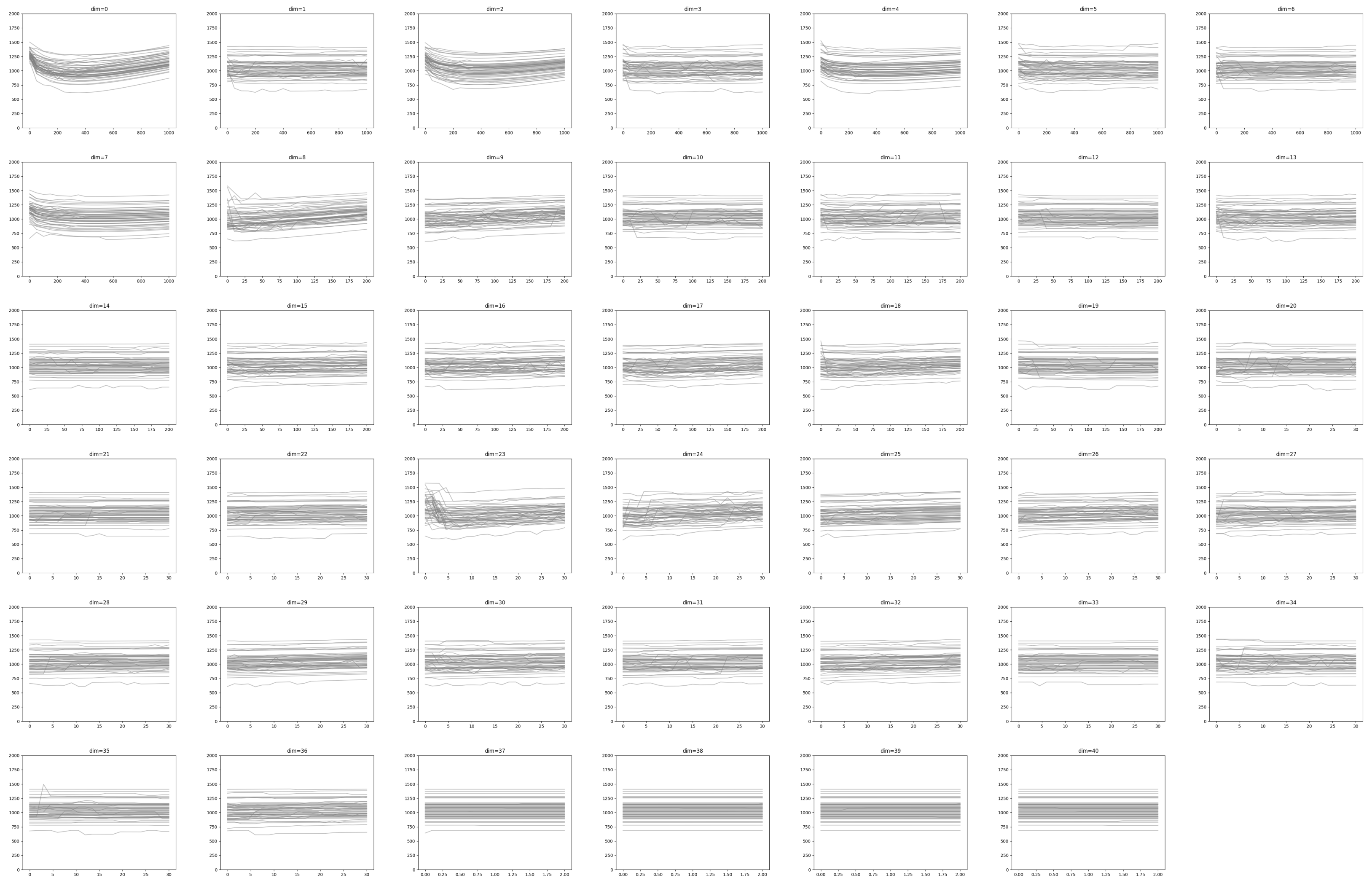


Figure 4: Im1H

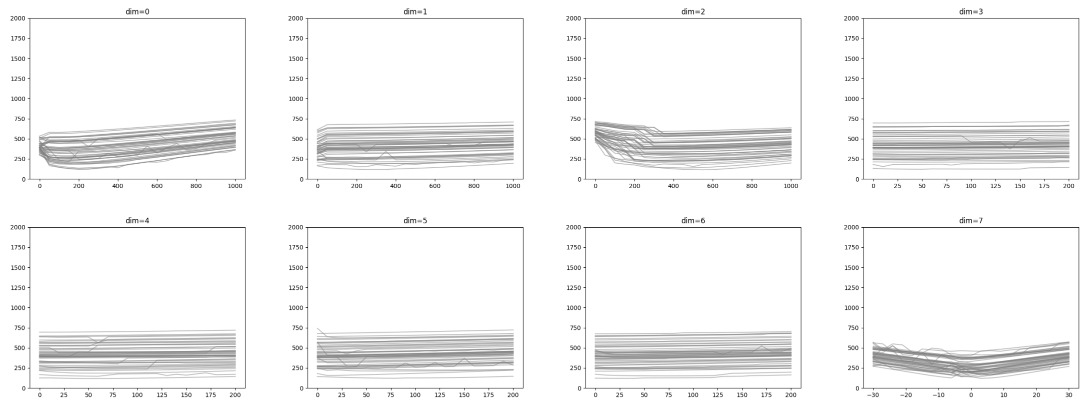


Figure 5: OTf- (CLaP)

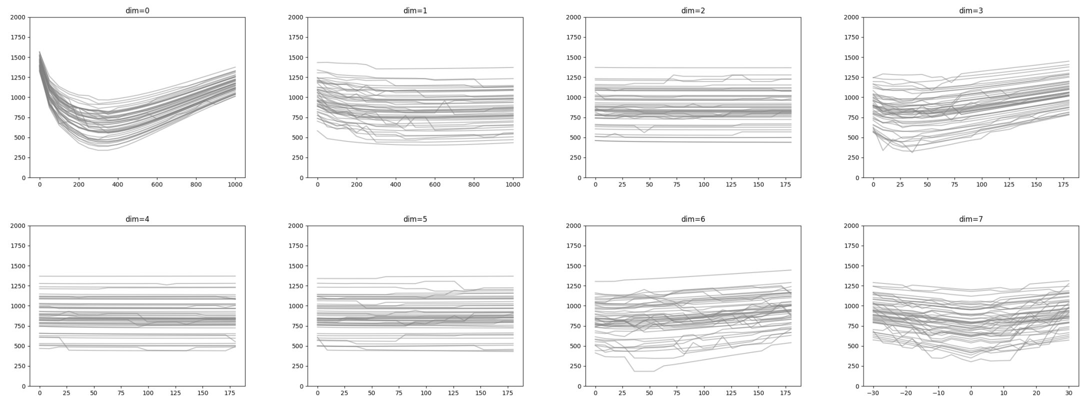


Figure 6: OAc (CLaP)

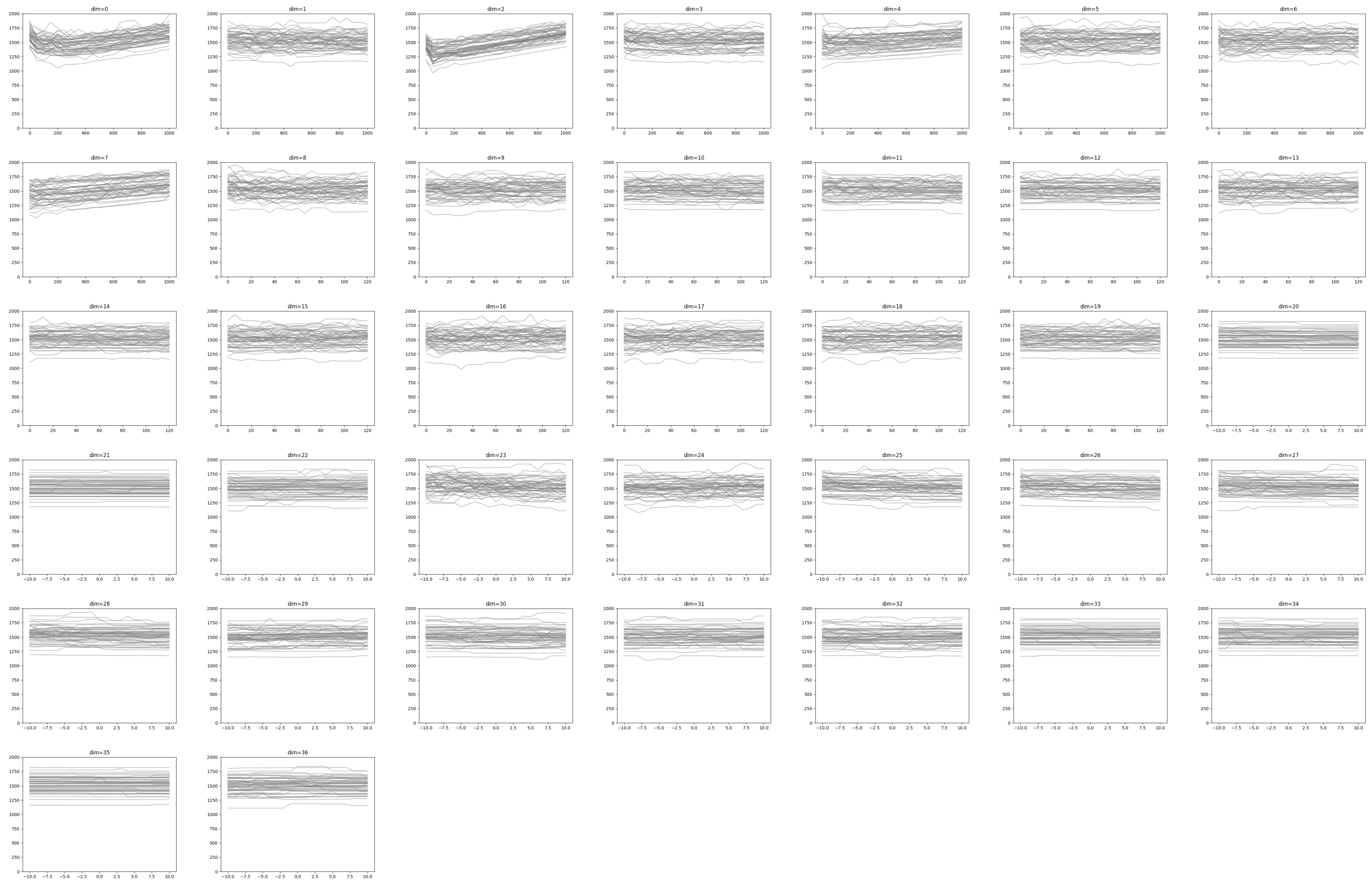


Figure 7: Im1H (CLaP)

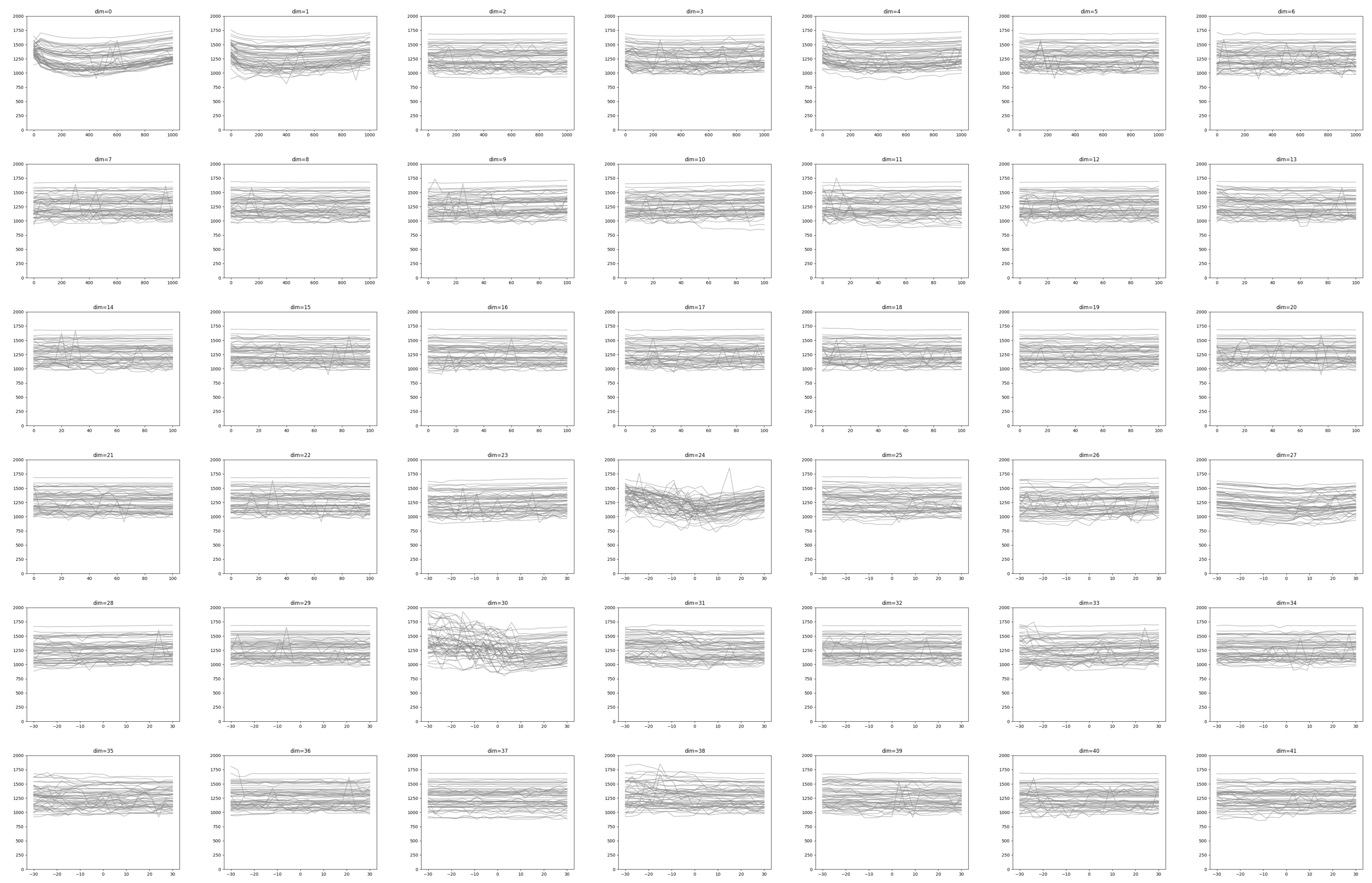


Figure 8: Im21 (CLaP)