Reconstructing crystal structures from global SOAP descriptors

Louis Adrian Böhm
Matrikelnummer: 608790

Zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

Betreuerin:
Prof. Dr. Dr. h.c. Claudia Draxl

Gutachter/Innen:
Prof. Dr. Dr. h.c. Claudia Draxl,
Prof. Christoph Koch, PhD

Eingereicht bei:
Institut für Physik, Humboldt-Universität zu Berlin

Eingereicht am:
12.07.2024

1 Abstract

A key driver for the tremendous advancements made in the application of machine learning in condensed matter physics and materials science is the use of descriptors to represent atomic environments [1, 2]. Such descriptors encode all relevant information about a system in a way that can enhance the performance and capabilities of machine learning models. While descriptors are widely used in models that predict material properties, their use in generative models is less common. Generative models enable the automated discovery of new materials with specifically tailored properties by extrapolating from known data [2]. To leverage the potential of descriptors in them, the model's are trained to predict descriptors from given properties. The desired structure is then retrieved by inverting the predicted descriptor back to the cartesian representation [3]. However, representations of crystal structures are often non-invertible, making the creation of generative models challenging [2]. An example for a descriptor that is widely used but can not be inverted analytically is the Smooth Overlap of Atomic Positions (SOAP) descriptor [4]. This work, presents a statistical inversion algorithm that can reconstruct the cartesian representation of crystal structures solely based on the SOAP descriptor. To fulfill this task, the Python program FUCrIMODo was written. It enables the creation of an algorithm that chains multiple Genetic Algorithms together to form a highly adjustable and generalizable workflow. A configuration of the program was created that is able to reconstruct the crystal structure of $SnCSi_3$ with a high degree of accuracy while still being designed to be applicable to a wide range of other crystal structures as well. This general applicability was then tested by applying it to a set of 47 other crystal structures of which 21 reached a high enough similarity to be considered successfully reconstructed. For nearly all structures it found the correct stoichiometry and atomic density. To emphasize the potential of the presented methods, a second configuration of the program was specifically adjusted to reconstruct the clathrate structure $Al_{16}Ba_8Si_{30}$. The reconstruction was done successfully, which shows how a strategic approach can be used to improve the algorithm and make it more general. No information about the target crystal structure, other than its descriptor was used during this process.

Contents

2 Introduction

Condensed-matter physics is the field that studies solid and liquid materials and aims to understand how their properties arise from their underlying structure and chemical composition. Important technological advancements, like the development of the transistor are the direct result of the research in this field [5]. In recent years, Machine Learning (ML) has gained significant popularity in computational condensed-matter physics [2]. High-accuracy ML models can be used to speed up materials' discovery by reducing the need for computationally costly quantum-mechanical calculations to predict the properties of potential new materials [6, 7]. The use of descriptors to represent atomic environments is a key driver for the ever-increasing use and success of ML [1]. A descriptor encodes all relevant information of a system in a numerical form which can be used as input for a ML model. Their ability to also hold information about the symmetries and invariances of a system, mitigates the need for a model to learn these properties from the data itself [4]. In practice, a ML algorithm is trained to model the relationship between descriptors and properties of known structures, which in turn enables these models to predict the properties of unknown materials [6]. Typical examples of such models are the kernel ridge regression or neural networks [8]. However, as demonstrated in Ref. [9] and [3], ML also holds potential for generative models, which aim to predict new structures based on a given property. To fully utilize descriptors in such models, it is necessary to retrieve the original Cartesian representation of a system from the descriptor by inversion. One option to address this challenge is the usage of invertible descriptors, such as SMILES [3, 9], which is, however, so far limited to the application for molecules. Reference [10] introduces the SPLICES descriptor which is specifically designed to be invertible but can only be used to represent crystal structures. The Smooth Overlap of Atomic Positions (SOAP) descriptor on the other hand, is more general and can represent molecules, surfaces, and crystals [4]. Due to its flexibility and accuracy it is widely used in models that predict material properties [11]. However, the SOAP descriptor cannot be inverted analytically, and currently no methods exist to globally reconstruct the Cartesian representation of the underlying structure [11]. Therefore, to leverage the full potential of the widely used SOAP descriptor in generative models, an inversion method is needed.

Two different methods for a local inversion have been reported in the literature. The first method is the usage of a gradient-based optimization algorithm that is able to recover atomic clusters after modifying their atomic coordinates. This is done by minimizing the differences between the descriptor of the modified cluster and the unmodified one, as a function of the atomic positions [4]. While this presents a method to perform a successful inversion, it can only be used if the structures used to initialize the algorithm have the same composition as the target and only slightly different atomic positions [4]. Reference [2] also introduces a local inversion algorithm with the same goal. The authors use a similar gradient-based optimization algorithm that is applied to structures with the same composition and slightly different positions than the target. In that work, the algorithm was successfully applied to molecules represented by the so-called Bispectrum descriptor, but the authors note that their findings are also extendable to SOAP. Regarding its applicability to crystals they note that more work is needed, but it should be possible. Similar to the first inversion method, the initial structures need to be similar to the target to achieve a successful inversion [2].

In this work, a more general inversion algorithm is reported that can recover the Cartesian representation of a crystal structure from its global SOAP descriptor. It is initialized with a set of randomly generated structures, eliminating the need of knowing the composition and atomic positions of the target. A Genetic Algorithm (GA), which is an optimization method inspired by the process of natural selection, is used for this task. The idea is to modify and recombine a set of crystals to test whether structural modifications increase the similarity of their descriptor to a given target descriptor. The ones with the highest similarity are then selected, and the process is repeated. This creates structures that are increasingly similar to the target. To implement the GA in this thesis, the Python program FUCrIMODO (Find unknown Crystal Structures by inversion of machine learning optimized descriptors) was created. It can be used to design highly customizable GA workflows by chaining multiple GAs together, each with a different purpose. This extends the idea of a cascaded GA proposed in Ref. [12] and *Goldberg's messy GA* [13].

The thesis starts with an introduction of the physical and computational concepts used to create the final algorithm. Then, an overview of FUCrIMODO is provided, followed by a detailed description of the GA configuration used to retrieve the crystal structure of $SnCSi_3$ from its SOAP descriptor, with a high precision. Next, the GA is applied to 47 other crystals to demonstrate that the GA design is not specific to one crystal. A second configuration of the GA is then introduced, which is tested on clathrate $Al_{16}Ba_8Si_{30}$, a complex structure with a large unit cell, to show how the algorithm can be adjusted to improve its generality.

3 Crystals and Crystal Structures

Crystals are homogeneous solid materials whose atoms are arranged in highly ordered, repeating patterns. The study of a crystal's internal structure is essential for understanding its chemical and physical properties [14]. The following explanation is based on the book *Crystals and Crystal Structures* by R. J. D. Tilley [14].

The underlying mathematical structure, that is used to describe a crystal is the *crystal lattice.* It is infinite and consists of periodically arranged *lattice points.* Each point can be defined, relative to a chosen origin, by the vector $\mathbf{r}(u, v, w)$.

$$\mathbf{r} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c} \tag{1}$$

Here, $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are the *basis vectors*, and u, v and w are integer coefficients, used to describe every lattice point. Their norms $|\mathbf{a}| = a$, $|\mathbf{b}| = b$ and $|\mathbf{c}| = c$, are called the *lattice constants.* The angles between the vectors are denoted with α, β and γ. Lastly, the angles between a vector and the plane spanned by the other two vectors is symbolized as ϕ, χ and ψ [15]. On the left side of Fig. 1, a finite representation of a three-dimensional crystal lattice is depicted. The lattice points and four sets of possible basis vectors are indicated using different colors. All of these sets can be used to describe the whole lattice, showing that the choice of basis vectors is not unique. As seen in the figure, each set of basis vectors spans a box, representing the *unit cell.* The unit cell is a parallelepiped with volume

$$V = |\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})| \tag{2}$$

By periodically stacking exact copies of the unit cell in all three dimensions, the whole crystal is formed. Since the unit cell is defined by the basis vectors, it is not unique. To ease the comparisons between different structures and to simplify calculations, it should be chosen according to a convention. A commonly used convention is the *primitive unit cell.* It only contains a single lattice point, making it the smallest possible unit cell, and hence it is preferred in numerical calculations, e.g. density functional theory which scales with the third power of the number of atoms. While its definition is simple, it is not always the most convenient choice. The *conventional unit cell* is chosen to reflect the symmetry of the structure and to simplify its mathematical description which makes analytical calculations easier. To form the crystal structure, each lattice point is populated with the *basis*, which consists of a fixed number of atoms, ions, or molecules. The right side of Fig. 1 shows a finite representation of a lattice and a basis consisting of two atoms next to it. The crystal structure on the right is formed by placing exact copies of the basis on each lattice point.

In this thesis, the Python library *Atomic Simulation Environment* (ASE) [15] is used to represent, manipulate, store, and visualize crystal structures. It provides a Python object, called `Atoms` to store the atomic positions, species, unit cell, and other properties of a crystal structure. This enables a consistent and easy way to work with crystal structures in Python.

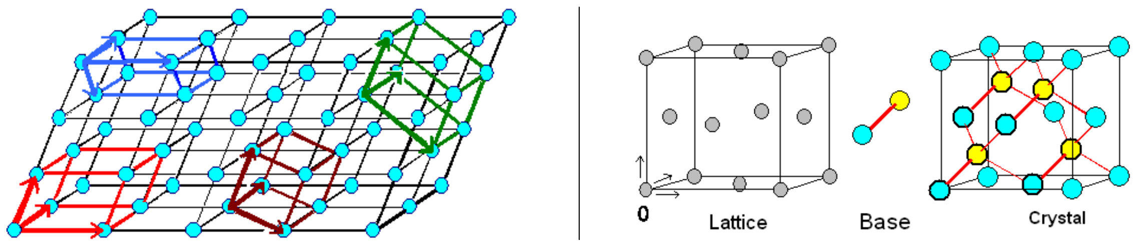


Figure 1: *Left:* Representation of a crystal lattice. The light blue dots represent the lattice points, and four different choices of basis vectors are displayed in red, green, yellow and blue, respectively. The unit cells, spanned by the respective set of basis vectors, are shown in the same color. *Right:* Example of a crystal lattice (left), where the lattice points are colored in gray. The basis on the right consists of two atoms, colored in blue and yellow, respectively. Next to it, the crystal structure is shown, which is formed by placing the basis on the lattice points. Source [16].

4 Methodology

4.1 Descriptors

To enable the efficient use of Machine Learning (ML) models in the field of computational materials sciences, atomic environments, like crystal structures, need to be transformed into a computationally manageable form, like a vector or an array. These forms are called *descriptors*, and their ideal properties as well as short descriptions of them are discussed in the following [6]:

- Invariance w.r.t. spatial translations and rotations of the coordinate system:
  Since the choice of basis vectors for a crystal structure is not unique and does not affect the properties of the crystal, the descriptor should not depend on them.
- Invariance w.r.t. permutations of the atomic indices:
  The order in which the atoms are listed in the basis does not affect the properties of the crystal structure, therefore the descriptor should not depend on it.
- Unique construction for each unique atomic environment and property:
  To consistently represent and compare different atomic environments, descriptors should be unique for each of them and should also only correspond to a single property.
- Continuous:
  Descriptors should be continuous to ensure that they reflect even small changes in the atomic environment, enabling the ML model to differentiate between them.
- Compact:
  To make efficient and accurate predictions, the descriptor should only contain information about the atomic environment that is necessary for the ML model.
- Computationally cheap:
  A descriptor is only useful in practice, if its construction and use in an ML model is computationally cheaper than the calculations or methods it aims to replace.

While all of these properties define the ideal descriptor, not every implementation has to fulfill all of them [4]. However, they ensure that ML models, trained on descriptors, respect the symmetries of a crystal system without the need to learn them explicitly. Additionally, such descriptors enable the models to precisely distinguish between structures, which is crucial for making accurate predictions [6]. Descriptors used to represent chemical environments are typically either *local* or *global*. A local descriptor describes localized regions of a structure and can be used to predict properties like atomic forces or absorption energies. Global descriptors, on the other hand, describe the whole structure, enabling the prediction of properties such as formation energies or band gaps [6].

SOAP Descriptors

The Smooth Overlap of Atomic Positions (SOAP) descriptor can efficiently represent the local atomic environment of many different classes of materials and is invariant w.r.t. rotation, translation, and permutation [4]. For these reasons, it is widely used in the field of computational materials science and condensed matter physics [2, 11] The following derivation of it is based on Ref. [6].

To calculate the SOAP descriptor, the atomic density field $\rho^Z(\mathbf{r})$ is computed for each species Z.

$$\rho^Z(\mathbf{r}) = \sum_{i}^{|Z|} e^{-\frac{1}{2\sigma^2}|\mathbf{r}-\mathbf{R}_i|^2} \tag{3}$$

The index i sums over all atoms with atomic number Z and position $\mathbf{R}_i$. These calculations are performed up to a cutoff radius R_{cut} and if present, periodic boundary conditions are taken into account. The origin of the coordinate system is set to points of interest, typically an atomic position in the given unit cell. The local region of the density is then expanded in terms of the spherical harmonics $Y_{lm}(\theta, \phi)$ and the radial basis functions $g_n(r)$.

$$\rho^Z(\mathbf{r}) = \sum_{nlm} c^Z_{nlm} g_n(r) Y_{lm}(\theta, \phi) \tag{4}$$

Here, the coefficients c^Z_{nlm} are defined by the following equation:

$$c^Z_{nlm} = \int\int\int_{\mathbb{R}^3} dV \rho(\mathbf{r})^Z g_n(r) Y_{lm}(\theta, \phi) \tag{5}$$

From this, the rotationally invariant power spectrum vector $\mathbf{p}$ is calculated. Its elements are defined for all unique combinations of the species Z_1 and Z_2 and radial degrees n and n' up to n_{max}:

$$p^{Z_1,Z_2}_{nn'l} = \pi\sqrt{\frac{8}{2l+1}} \sum_m (c^{Z_1}_{nlm})^* (c^{Z_2}_{n'lm}) \tag{6}$$

The angular degree is calculated up to l_{max}. As mentioned before, the calculation is typically done for every atom in the unit cell, meaning that more than one descriptor is retrieved. By averaging over all of these local descriptors, a single global descriptor is obtained. It is independent of the number of atoms and can therefore be used to compare structures of different sizes.

All SOAP descriptors in this thesis are calculated with the Python library *DScribe* [6] [17]. The descriptors provided by DScribe are highly customizable and can be adjusted to fit the specific needs of the task at hand.

Definition 1 (SOAP Descriptor $\mathbf{d}$)**.** In the following, the SOAP descriptor of a crystal structure, calculated with DScribe and the parameters shown in Tab. 1, is denoted as $\mathbf{d}$.

The SOAP descriptor $\mathbf{d}$ can be written as a vector of length 2700, where each entry is a part of the averaged power spectrum obtained with:

$$p_{nn'l}^{Z_1,Z_2} \propto \sum_m \left(\frac{1}{N_\mathrm{A}} \sum_i c_{nlm}^{i,Z_1}\right)^* \left(\frac{1}{N_\mathrm{A}} \sum_i c_{n'lm}^{i,Z_2}\right) \tag{7}$$

The coefficients c_{nlm}^{i,Z_1} and $(c_{nlm}^{i,Z_2})^*$ are calculated for all atoms i in the unit cell and are then normalized by the total number of atoms N_A in the cell. The entries in the output vector are ordered with respect to Z_1, Z_2, n, n', and l, to ensure that descriptors can be consistently created and compared. Figure 2 illustrates the averaged SOAP descriptor of a $\mathrm{SnCSi_3}$ crystal structure[1]. The parts of the descriptor corresponding to the interactions between atoms of species Z_1 and Z_2 are separated by a dashed line. By only observing a species-specific subset of a SOAP descriptor, statements about this species' environment can be made. This is relevant for the optimization process of the GA (see Section 5.2), and will be explained in more detail later.

Parameter	species	r_{cut} [Å]	n_{max}	l_{max}	sigma	periodic	average
Value	C, Si, Ge	15.0	8	8	0.5	True	inner

Table 1: Parameters used to calculate all SOAP descriptors in this work. The parameter set for the species is only an example and must be adjusted to the specific task.

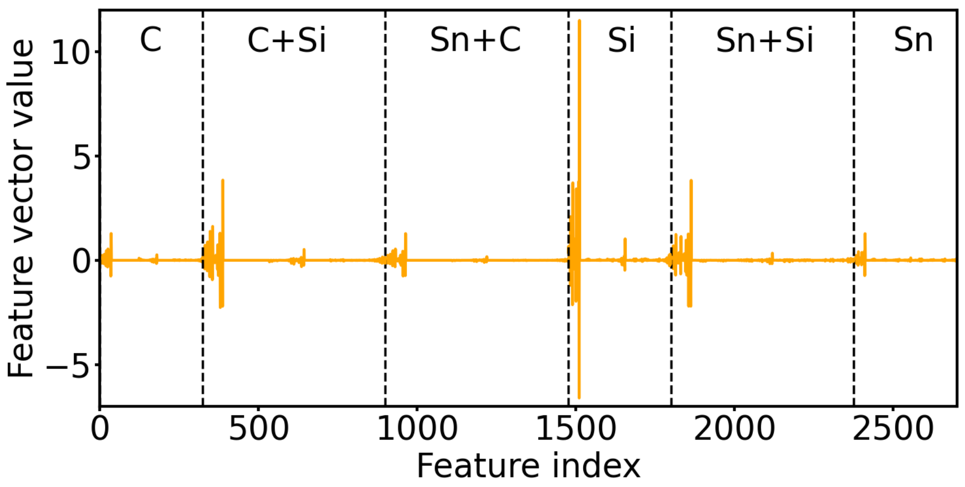


Figure 2: Values of each entry in the averaged SOAP descriptor of $\mathrm{SnCSi_3}$ [18]. Dotted lines separate the sections of the SOAP descriptor representing different species. The corresponding species combinations are written in the top of each section.

[1]The crystal structure is part of the Ternary Dataset (see Section 4.3).

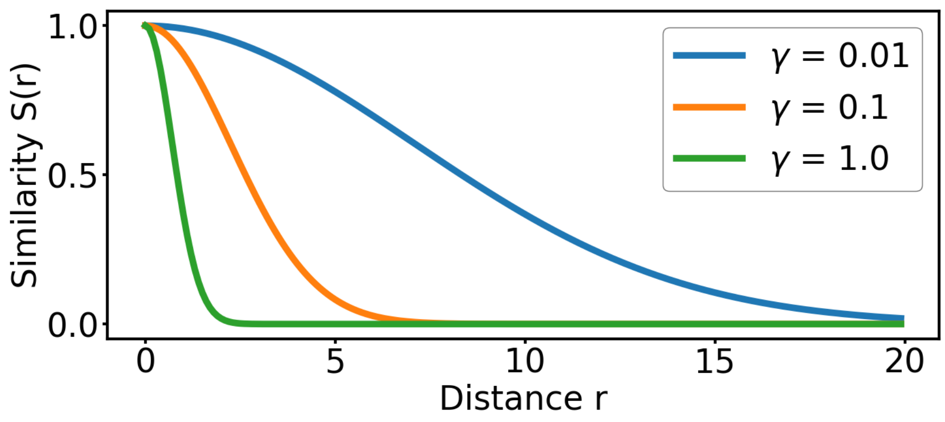


Figure 3: RBF similarity S(r) as a function of the distance r for different values of the parameter $\gamma \in \{0.01, 0.1, 1.0\}$.

Similarity

To consistently measure how similar two crystal structures are, their SOAP descriptors, $\mathbf{d}_1$ and $\mathbf{d}_2$, are compared with the Radial Basis Function (RBF) similarity $\mathrm{S}(\mathbf{d}_1, \mathbf{d}_2)$ [4]. In this work, the similarity is used to compare the descriptor $\mathbf{d}$ of a crystal structure to a fixed target descriptor $\mathbf{d}_{\mathrm{tar}}$. Following the implementation of the RBF similarity in the Python library Scikit-learn [19], the similarity to the target $\mathrm{S}(\mathbf{d}, \mathbf{d}_{\mathrm{tar}})$ is defined as:

$$\mathrm{S}\left(\mathbf{d}, \mathbf{d}_{\mathrm{tar}}\right) = \exp(-\gamma \cdot ||\mathbf{d} - \mathbf{d}_{tar}||^2) \tag{8}$$

The hyper parameter $\gamma > 0$ must be adjusted to the task at hand. Figure 3 illustrates how the choice of γ affects the similarity S(r) as a function of the distance $\mathrm{r} = ||\mathbf{d} - \mathbf{d}_{\mathrm{tar}}||$. It can be observed that for all $\gamma \in \{0.01, 0.1, 1\}$, the similarity decreases exponentially with increasing distance, but at different rates. For $\gamma = 1$, the similarity approaches 0 for distances greater than 3; for $\gamma = 0.1$, the same occurs at distances above 6; and for $\gamma = 0.01$, it happens at distances above 20. Beyond these thresholds, even vastly different distance values cannot be easily distinguished anymore, since they all have similarity values close to 0. If these similarities differ by values smaller than the numerical precision of the computer, they can not be distinguished at all. Therefore, when distances between descriptors and the target are expected to be large, a small γ is preferred to ensure that they can be properly compared. However, as shown in the graph for $\gamma = 0.01$, the slope of the similarity curve flattens for distances close to 0. In this chase, a bigger γ is preferable, since it has a steeper slope near distances close to 0, as shown in the graph for $\gamma = 1$. To enable the comparison of vastly different structures, while still being able to distinguish between similar ones, in this work multiple γ values are used simultaneously. The following definition is made to ensure that calculations are comparable.

Definition 2 (Reference similarity $\mathrm{S_r}$)**.** The *reference similarity* $\mathrm{S_r}$ is defined as the RBF similarity of a crystal structure's SOAP descriptor $\mathbf{d}$ to the target SOAP descriptor $\mathbf{d}_{\mathrm{tar}}$ with a γ-value of 0.1. All results in this work are evaluated with this reference similarity.

Additional Descriptors and Similarities

Since the algorithm developed in this work will optimize structures based on the RBF similarity with the SOAP descriptor, the results must be confirmed with additional descriptors and similarity measures. This is done to avoid biases that could be introduced by the implementation of the SOAP descriptor or the similarity measure. The first descriptor used for this is an ordered list of the distances and angles between atoms in the unit cell. This descriptor does not depend on the choice of basis vectors and is therefore invariant w.r.t. spatial translations and rotations of the coordinate system but can only be used to compare structures with the same composition. Furthermore, it is not invariant w.r.t. permutations of the atomic indices and must therefore be sorted to ensure that the correct atoms are compared. The arithmetic mean of the absolute differences between the angles and distances can then be used to measure how different the structures are. Additionally, the Radial Distribution Function (RDF) is also used as a descriptor. The RDF describes the distribution of atoms around a reference atom [20]. It only depends on interatomic distances, and is therefore invariant w.r.t. spatial translations and rotations of the coordinate system. Furthermore, it is also invariant w.r.t. permutations of the atomic indices, which makes it better suited to compare complex structures that mayo not be visually similar. The RDF is calculated with the implementation from the Python library ASE [15] as a histogram with N_b bins:

$$h_{r_n} = \frac{2\pi \cdot N_{\mathrm{A}}^2}{V} \cdot dr \sum_{i \neq j} \delta(\lceil \frac{|r_j - r_i|}{dr} \rceil - n) \tag{9}$$

The n-th bin describes the distribution of atoms h_{r_n} close to the distance $r_n = dr \cdot n$, where $dr = r_{\mathrm{max}}/N_b$ is the width of the bins and r_{max} is the defined maximum distance that is considered. The first factor of the equation is only a normalization constant, calculated with the number of atoms N_{A} and the volume of the unit cell V. Then all distances between the atoms i and j at the respective positions r_i and r_j are divided by dr and rounded to the next bigger integer, with the rounding operator $\lceil ... \rceil$. The Kronecker delta function δ is then used to only count those atoms that are close to the distance r_n and should therefore be in the n-th bin. This is done for each of the N_b bins to create an RDF histogram that is then used to compare structures qualitatively.

4.2 Genetic Algorithm

Genetic Algorithms (GAs) are an optimization technique inspired by Darwin's theory of evolution [21]. They are used in a variety of applications, where they are able to find solutions to complex problems [11, 12, 22]. All following explanations of the general approach to GAs is based on the book [21]. Additional references will be provided where necessary. For the purpose of this thesis, a GA is used to find an unknown *target* crystal structure that is the inverse of a given SOAP descriptor, called *target descriptor* $\mathbf{d}_{\text{tar}}$. This is achieved by repeatedly modifying and recombining a set of crystal structures with the objective of increasing their similarity to the target. These structures form the *population*, where each structure in it is called an *individual.* A population that has gone through a complete iteration of the optimization process is called a *generation.* Figure 4 shows a typical workflow of a GA. The algorithm starts with an initial population consisting of randomly generated structures. First, an *evaluation* is performed, which measures key qualities of each structure to make them comparable to each other. Next, the *genetic operators* select, modify, and recombine a subset of the population to create new structures. In the last step, the *survivor selection* decides which structures are passed on to the next generation. This process repeats until the algorithm reaches a break condition. A more detailed explanation of the individual steps will be given after a brief overview of the specific GA implementation used in this work.

The GA developed in this thesis consists of multiple *stages* that are chained together to form the workflow shown in Fig. 5. The workflow starts with an initial population of crystal structures, which is then passed to the first stage. The stage selects a subset of the population and then uses a GA with specifically selected parameters to optimize it. The optimized population is then passed on, and the process repeats until no further stages are left. As mentioned in the introduction, this approach extends on the idea of a cascaded GA proposed in Ref. [12] and Goldberg's messy GA [13].

This multi-stage approach is applied in the following two ways: First, the algorithm chains together stages that allow for increasingly bigger unit cells, so that a variety of structures, within each unit cell size, can be explored before the algorithm moves on to the next size. Second, stages are chained together that alternate between exploration and refinement of the structures. In the exploration stage, the structures get heavily modified to strategically produce a wide range of possible structures and test whether they are similar to the target. The refinement stage focuses on optimizing the structures found during the exploration, by applying only small modifications. This ensures that the algorithm can explore a wide range of structures while still being able to carefully improve the best ones. An in-depth description of the stages will be given in Section 5.2. The following is a detailed explanation of the workflows shown in Figs. 4 and 5.

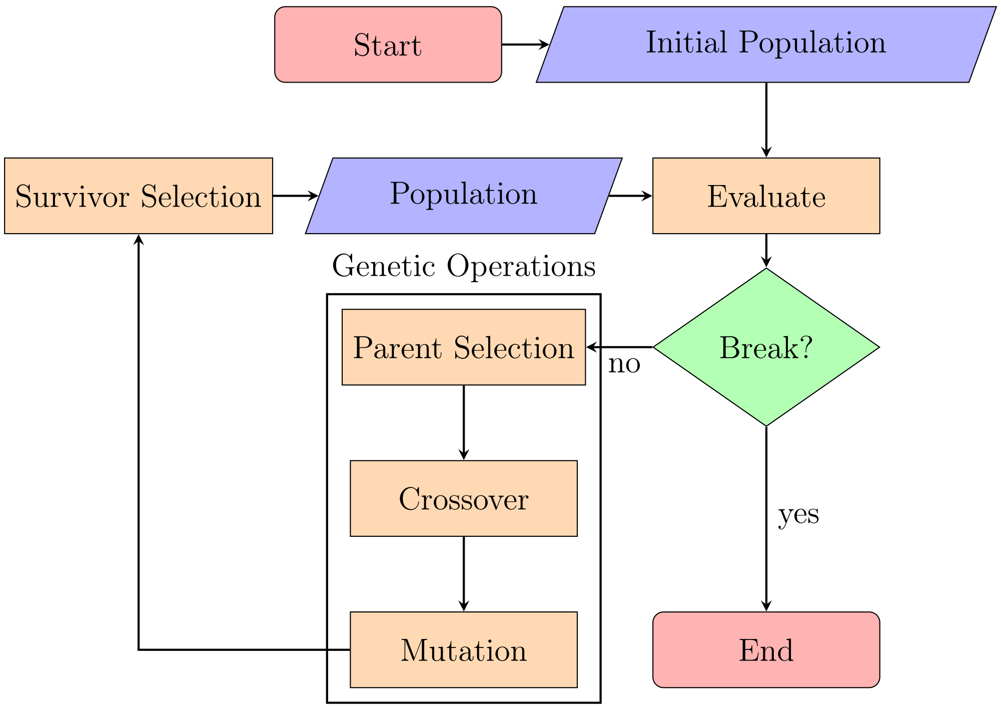


Figure 4: Example workflow of a genetic algorithm. The orange nodes symbolize operations that are applied to the population. The green node is a decision node and the blue nodes represent the population. The genetic operators are grouped together to show that they are is combined operation that is used to modify the population.

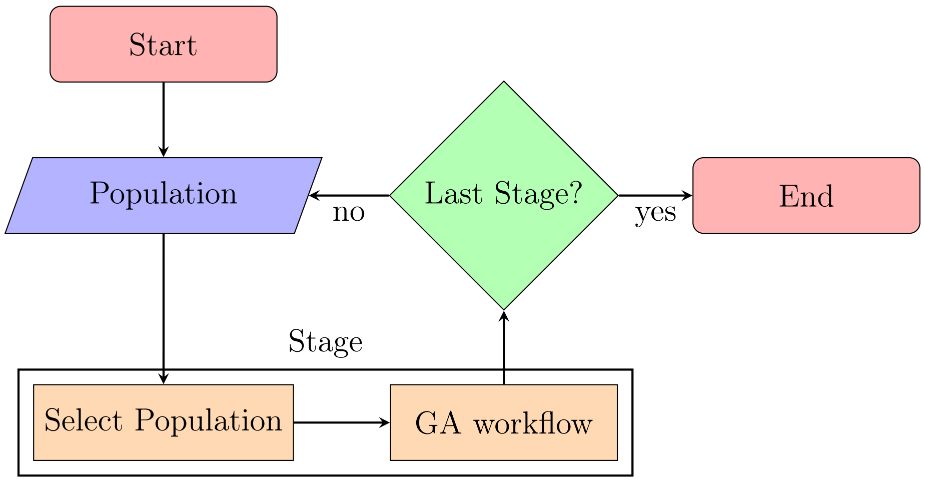


Figure 5: Workflow to visualize how the stages are applied in the optimization process. Each stage consists of a complete GA workflow, similar to the one shown in Fig. 4 and a process which selects the individuals for the new population from the population created in the previous stage. The algorithm stops after the last stage is completed.

Population Initialization

The GA starts with an initial population of random crystal structures that are created by placing atoms in unit cells with random cell vectors. The atomic species and maximal/minimal cell sizes used to create these structures are chosen for each problem individually.

Fitness Function

The population is then evaluated by a *fitness function*, which assigns a scalar value, called the *fitness*, to each member of the population. This value allows the GA to consistently compare individuals and distinguish between more and less optimal structures. Several considerations must be taken into account when designing an effective fitness function. Firstly, the fitness function should be continuous to ensure that even small improvements in an individual are detectable by the GA. Additionally, it should have a clear maximum and minimum to ensure that differences in the fitness value are meaningful and comparable. Multiple fitness functions can be used together, to simultaneously evaluate different aspects of an individual. In this chase each fitness function gets a *fitness weight* that determines how much it contributes to the overall fitness. The fitness functions created and used in this thesis are listed in Tab. 2.

Fitness name	Explanation
Physicality	Measures the distances between all atoms in the structure. If a distance is lower than a defined threshold, the fitness is decreased.
Similarity	Calculates the RBF similarity between the SOAP descriptor of the structure and the target SOAP descriptor. The γ parameter is adjustable.
Species Specific Similarity	Calculates the RBF similarity only for the species-specific parts of a structures SOAP descriptor and the target. (see Fig. 2)
Number of Atoms	Assigns a fitness value f based on the number of atoms N_A in the unit cell, according to the formula: $f = (\arctan(-\mathrm{N} + \mathrm{N}_\mathrm{max} + 3) + \pi/2)/\pi$, where N_max is the maximum number of atoms the structure should have.

Table 2: Fitness functions that are used in the GAs of this thesis. All functions are bounded between 0 and 1, where 1 is the best possible fitness value.

Break Conditions

After the evaluation of the fitness (see Fig. 4), the algorithm evaluates if its break condition is met. The break condition checks whether a maximum number of generations has been reached, if an individual with a desired fitness value has been found, or if the population cannot be further optimized. Each stage has its own break condition, to determine how the algorithm should proceed. With this strategy, the GA can be made very versatile by chaining stages that work well for one type of crystal structure, with stages, optimized for another type. To maintain efficiency, stages that are not useful for a given problem are then automatically skipped by using an appropriate break condition. Furthermore, if an exploration stage finds a structure that is already quite similar to the target, the algorithm moves on to the refinement stage. Operations in the refinement stage do not waste computational resources to find entirely new structures but only focus on the existing ones, which in turn speeds up the whole optimization process.

Genetic Operators

If the break condition is not fulfilled, the *genetic operators* of the current stage are applied to the population. This process typically begins with the *selection* operator, which picks individuals to which the *crossovers* and the *mutations* are applied to. The operators perform modifications on the crystal structures and the population.

Selection

There are two different types of selections in a GA. The first is called the *parent selection. Parents* are the individuals selected to be modified to form the *offspring.* The selection process is based on the fitness of the individuals and is the primary mechanism, which enables the GA to optimize the population towards the optimal solution. The extreme cases of selection are either selecting only the individuals with the highest fitness or selecting them randomly. The first case, called *elitism*, can help to optimize the population, but often leads to premature convergence into a local optimum. The second case, *random selection*, would not lead to any optimization, because it lacks selection pressure that is needed to improve the population. Therefore, the appropriate choice of the selection lies somewhere in between these two extremes and is crucial to the success of the GA. Typically, selection is done proportionally to the fitness of the individuals. In this thesis, the *tournament selection* from the DEAP library [23] is used. It selects $N_{\text{tour.}}$ random individuals from the population and chooses the individual with the highest fitness from this subset. The parameter $N_{\text{tour.}}$ is called the tournament size and must be adjusted appropriately. A smaller tournament size leads to a more random selection, while a bigger size tends towards elitism. The second selection, called the *Survivor Selection*, takes place after all genetic operators were applied. For this, the *NSGA-II* (non dominated sorting genetic algorithm II) selection, as proposed in Ref. [24], is used. Its design ensures that multiple fitness values are considered, while simultaneously keeping the population diverse and preserving its best individuals.

In the context of GAs, *Diversity* describe how much variation there is between individuals within a population. A high diversity indicates that individuals posses a wide range of characteristics and ensures that the algorithm does not get stuck in a local optimum. During the exploration stages of the GA, it should maintain a high diversity to find a wide range of possible structures. On the other hand, the diversity should be reduced during the refinement stages, since these should primarily focus on optimizing the best structures found during the exploration. The selection process directly influences the diversity of the population.

Crossover

The crossover operator is applied after the parent selection (see Fig. 4). The goal of this process is to combine the best aspects of two parents into a single offspring. It is crucial that all structures created through the crossover operation must conform to the physical constraints of the problem to avoid optimizing non-physical structures. The likelihood of applying a crossover on a pair of parents can be controlled by the *Crossover Probability*. If no crossover is applied, the offsprings are exact copies of their parents. The specific crossovers used in this work are listed in Table 3.

Name	Short Explanation
Exchange elements	A random subset of the species list from each parent is exchanged between them. The maximal length of this subset is determined by the size of the smaller parent.
Exchange cell vec.	A random number of unit cell vectors are exchanged between the parents. After this, the atomic positions are scaled to fit the unit cell dimensions.
Stack cells	The *stack* method of the ASE library [15] is used to combine the parents by placing one next to the other, to form a new composite structure.

Table 3: Explanation of crossovers used in this work.

Mutation

Mutations are applied after the crossovers have been performed. They modify aspects of an individual, such as atomic positions, cell size, cell shape, or the species to form a new structure called *Mutant*. Similar to the crossovers, the mutation must create physically possible structures. The *Mutation Probability* controls the likelihood that a mutation is applied to an individual. The specific mutations used in this work can be found in Tab. 4.

Name	Short Explanation
Permute*	Atoms in the structure are randomly exchanged with each other.
Rattle*	The positions of a few atoms are randomly but only slightly changed.
Add	Additional atoms are randomly placed in the unit cell of the structure. The species of the new atoms can be adjusted.
Delete	A random atom is removed from the structure.
Replace	The species of a randomly chosen atom is replaced with another species. The allowed species can be adjusted.
Strain*	The cell vectors of the structure are randomly stretched or compressed, and the atomic positions are scaled to fit the new cell dimensions.
Soft Mutate*	Mutates the structure by displacing it along the lowest (nonzero) frequency mode found by vibrational analysis. [15]
Cutout	Shrinks the unit cell and deletes all atoms outside the new cell dimensions.
Minimize tilt	Uses the ase.build.minimize_tilt method [15] to minimize the angles between cell vectors, while keeping the cell volume constant.
Convert to conv. cell	Converts the structure to its conventional cell using the MatId symmetry analyzer [25].
Enlarge	Duplicates the structure and places the copy next to the original to form a new composite structure.
Rotate*	Rotates all atomic positions around a single point.
Multi	Chains multiple mutations together in a random or predefined order.

Table 4: Explanation of the mutations created and used in the thesis. The mutations marked with * are taken from the ASE library [15] and have been slightly modified.

Adjusting the GA

In addition to the genetic operators and the fitness function, there are numerous other aspects that need to be adjusted. These include, for example, the number of individuals created in each generation or the way the genetic operators are applied. The workflow shown in Fig. 4 is therefore merely an example of how a GA can be structured. The specific implementation can vary greatly depending on the problem at hand and many different configurations must be tested to find the optimal one. To evaluate whether a GA performed well in this work, the following definition is made:

Definition 3 (Maximum similarity $\mathrm{S_{max}}$)**.** Let $\mathbf{D}$ be the set of SOAP descriptors of all crystal structures that ever existed in the population of a given GA. The *maximum similarity* $\mathrm{S_{max}}$ is defined as:

$$\mathrm{S_{max}} = \max(\{\mathrm{S_r}(\mathbf{d}, \mathbf{d}_{\mathrm{tar}}) \,|\mathbf{d} \in \mathbf{D}\}).$$

4.3 Datasets

In this work, the crystal structures from three different datasets are used. The first is the *Ternary* dataset which was introduced in Ref. [26]. This dataset was created to test a novel method of identifying suitable descriptors for the prediction of material properties. It was created with *ab initio* calculations and contains 589 crystals in total, which are mostly ternary compounds but also include some binary and unary crystals. All data are available at the NOMAD repository with the link provided in Ref. [18].

Additionally, crystal structures from the *Kaggle-NOMAD* dataset are used, which was part of the competition discussed in Ref. [8]. The objective in the competition was to create a ML model that can predict the formation and band-gap energies of the crystals in the dataset. It contains 3,000 $(\mathrm{Al_xGa_yIn_z})_2\mathrm{O_3}$ compounds (with $\mathrm{x+y+z} = 1$), each having 10 to 80 atoms in the unit cell.

Furthermore, the *MP* dataset is used, which is a subset of a larger database, selected to only contain structures of which the properties were calculated with a specific *ab initio* method. It is introduced in Ref. [27] and was used to train a message passing neural network to classify structures as metallic, insulating, or semiconducting and to predict band-gap energies of non-metallic structures. The dataset contains 46,090 metals and 16,012 non-metals.

Finally, the crystal structure of $\mathrm{Al_{16}Ba_8Si_{30}}$ is taken from the dataset introduced in Ref. [28]. All materials in the dataset belong to the class of clathrates. Clathrates consist of cages in which guest atoms can be trapped to form structures with a low thermal conductivity while maintaining a high electrical conductivity, or in some cases, also semiconducting properties [28].

5 Results

5.1 Implementation

I wrote the Python program FUCrIMODo (Find Unknown Crystal by Inversion of ML Optimized Descriptors), to design and run all GA workflows described in this work. The program has all the functionality needed to run and analyze a GA that aims to approximate the Cartesian representation of a crystal structure, solely based on the structure's descriptor. While it is not limited to a specific descriptor, it was optimized and tested only with SOAP descriptors. The configuration of a FUCrIMODo GA workflow is done via a Python script, enabling the use of additional Python libraries and tools to create complex setups. These scripts are from now on referred to as *GA configurations* or *program configurations*. To enable the easy creation and adjustment of such scripts, FuCrIMODo follows the idea of modular design, making all parts of the GA easily interchangeable. This is achieved by using abstract classes[2], which provide basic functionality and a predefined structure for the classes that inherit from them. Figure 6 shows how the mutation and crossover classes use inheritance to ensure the correct creation of new structures. The input of the workflow is a crystal structure which is then manipulated by the user defined modification operations. The manipulated offspring is then validated by first checking whether instead of an `ase.Atoms` object, a `NaN` value was returned. After that it checks if the SOAP object can be created from the returned object.

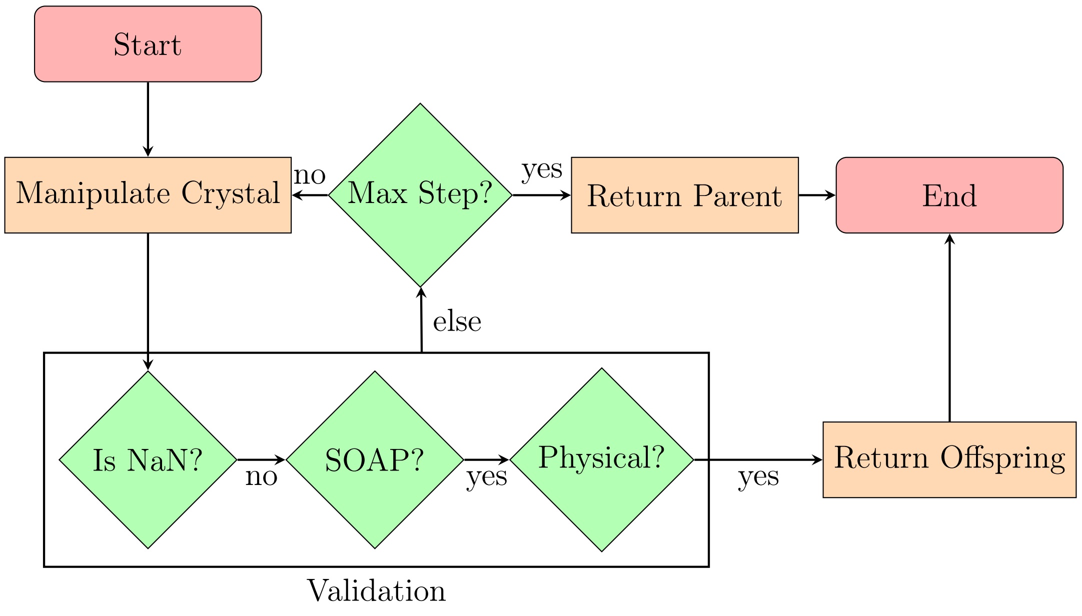
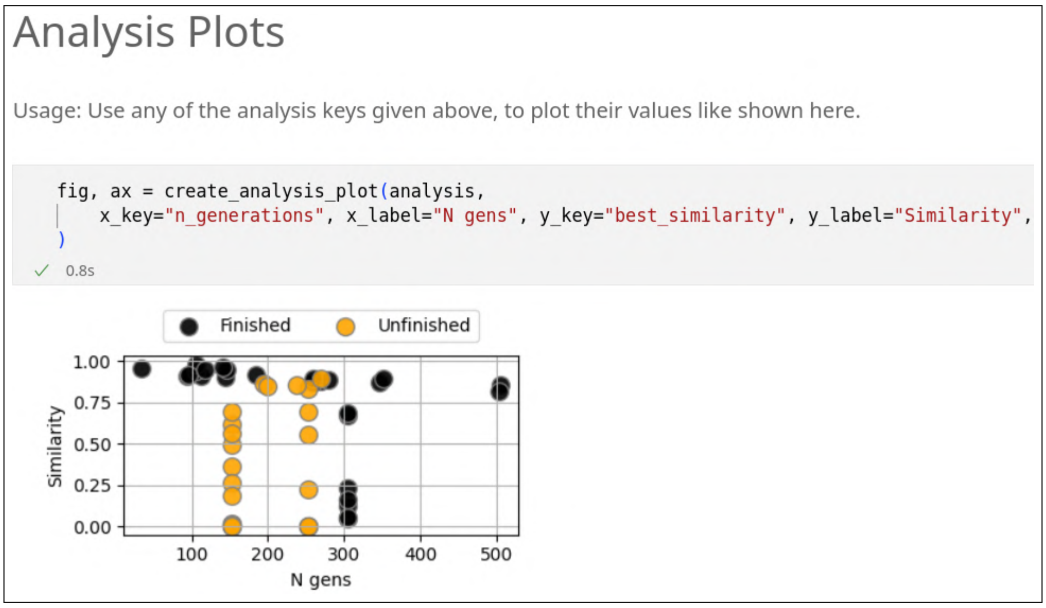


Figure 6: Workflow of the crossover and mutation classes in FUCrIMODo. The validation step is inherited from their respective parent classes and therefore grouped together in a box. Each validation step is a decision node, where in case the condition is not met, the manipulation of the crystal is repeated. This is done until a valid offspring is created or the maximum number of steps is reached.

[2]Abstract classes are a part of the Python ABC library [29] and are commonly used to create a framework for a specific task.

The final step validates that the created crystal is physical. Here, physicality is defined as the crystal structure having a large enough unit cell to fit all atoms, and that atoms are not to close to each other. Only if all these conditions are met, the offspring is considered valid and will be returned to the GA. If any of the validation steps fail, the process is repeated until a proper offspring is found or the maximum number of retries is reached. In the latter case, the class returns the parent structure to the GA and notifies it that no valid offspring could be found, so the GA can react accordingly. The reason for the failure is written to the log file, to allow for a detailed analysis of problems. Due to this design, there is no need of repeatedly implementing verification, which enables the easy implementation of new mutation and crossover operators.

The statistics tool from the DEAP library [23] is used to collect data during the run, which are stored in human-readable JSON files. Additionally, the best crystals that are found during the run are stored in an ASE database [15]. Since the analysis of a GA is crucial to find its optimal configuration, FUCrIMODo can automatically generate Jupyter notebooks [30] to ease this process. There are three levels of analysis that can be performed with these notebooks. The first is the analysis of a single run, the second is the comparison and analysis of multiple runs, and the third level is used for comparing and analyzing different sets of multiple runs. An excerpt of a level-two notebook is shown in Fig. 7. In it, the reference similarity S_r is plotted against the total number of generations each run took. By changing the parameters `x_key` and `y_key` to different values, all sorts of data can be visualized. The notebooks for the other levels are similar to this one, and many concepts can be reused between them to ensure a consistent analysis process. This provides a fast, easy, and flexible way to analyze results.


Analysis Plots

Usage: Use any of the analysis keys given above, to plot their values like shown here.

```
fig, ax = create_analysis_plot(analysis,
    x_key="n_generations", x_label="N gens", y_key="best_similarity", y_label="Similarity",
)
```
✓ 0.8s

Figure 7: Excerpt of an automatically generated Jupyter notebook that eases the analysis of the results of the GA.

5.2 Descriptor inversion

A GA workflow was created with FUCrIMODo. The following results show that this workflow is able to reconstruct the crystal structure of $SnCSi_3$ from its corresponding SOAP descriptor. Here, the successful reconstruction of the crystal structure is defined as reaching a maximum similarity S_{max} of 0.99. The workflow consist of four steps, which are called *stages* where the first two are used to find small structures that are potentially similar to the target, while the last two stages aim to find larger structures. In both cases, the first stage is designed to explore a wide range of structures, while in the respective second stage only the individuals found during the exploration are optimized. The initial population consists of 500 single atoms of species S, C or Si, each placed in a randomly generated unit cell that is limited by the *Bounds 1* shown in Tab. 8, in the appendix A.1. In all stages, the NSGA-II selection and the tournament selection, with tournament size $N_t = 4$, are used. Tables 10 and 9, in the appendix A.1, show additional parameters, operators, and break conditions that are set for each stage. The program was executed with this configuration, and the results of the completed run are analyzed with the tools provided by FUCrIMODo.

The upper panel of Fig. 8 shows the development of the reference similarity S_{r} (see Def. 2) for each generation. The minimum, maximum, and mean of S_{r} are shown in the top, the standard deviation in below it. In the bottom, these statistics are displayed for the development of the unit cell volume on a logarithmic scale. The vertical black lines separate the different stages. The respective numbers are displayed in the top. The horizontal dashed lines in the volume plot represent the maximum volume allowed for the structures created during the respective stage. For stage 1, the maximum volume is 512 Å^3, for stage 3, it is 2744 Å^3,[3] while stage 2 and 4 do not have a volume limit. The target structure has a volume of 500 Å^3 and is therefore within these limits. In the top of Fig. 9, the target structure is shown; below it, the best structures, meaning the ones associated with the maximum similarity S_{max} (see Def. 3), are depicted for each stage. The bond lengths and angles of the atoms in each structure, relative to the central C atom, are shown in the table at the bottom of the figure. In the following, these results will be discussed for each stage individually.

Stage 1: Exploration

The first stage aims to find a wide range of small structures, which could potentially be similar to the target. As discussed earlier the cell size is limited to 512 Å^3 and additionally the *number of atoms fitness function* (see Tab. 2) is used, to force the GA to mostly create structures with a small cell size and a low number of atoms. The similarity development in Fig. 8 shows that the maximal similarity (red line) first increases rapidly but starts to slow down after around 50 generations.

[3]Following Eq. 2 it can be easily shown that a unit cell reaches its maximum volume when its vectors are orthogonal to each other. With this, the volume can be simplified to $V_{max} = a \cdot b \cdot c$, where a, b, and c are the unit-cell parameters. For stage 1, the bounds 1 from Tab. 8 are used, which limit the volume to $V_{max} = (8\ \text{Å})^3 = 512\ \text{Å}^3$. Stage 3 uses the bounds 2, which allow for a maximum volume of $V_{max} = (14\ \text{Å})^3 = 2744\ \text{Å}^3$.

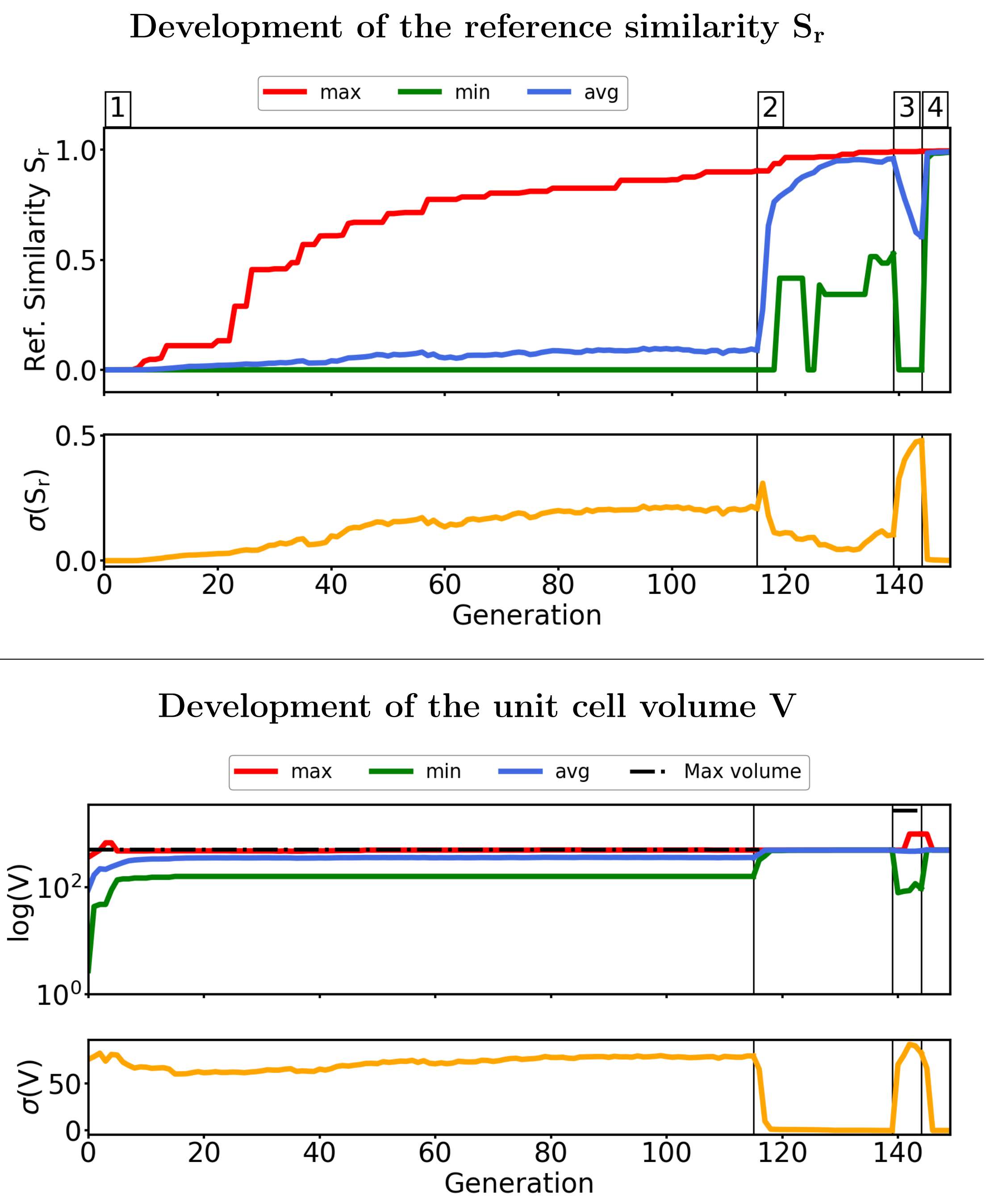
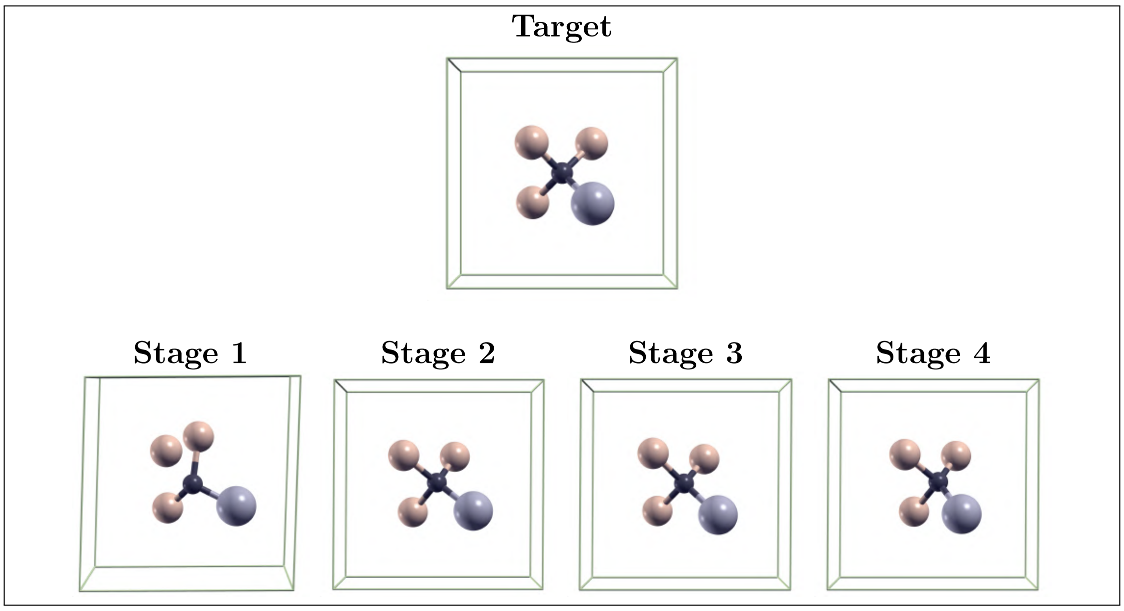


Figure 8: Development of the reference similarity S_r and the unit cell volume of the population for each stage of the GA discussed in this section. The maximum, minimum, and average for each generation are shown in the upper panels, the standard deviation in the bottom panels. The dotted, horizontal line in the volume development represents the upper limit set during stages 1 and 3.



Bond length [Å]

A1	A2	A3	Target	Stage 1	Stage 2	Stage 3	Stage 4
C	Si		1.91	1.93	1.73	1.73	1.73
C	Si		1.91	2.48	2.07	2.07	2.07
C	Si		1.91	1.66	1.94	1.94	1.93
C	Sn		1.91	1.93	1.97	1.97	1.96

Bond angles [°]

A1	A2	A3	Target	Stage 1	Stage 2	Stage 3	Stage 4
Si	C	Si	109.47	72.23	108.96	108.96	108.96
Si	C	Si	109.47	121.47	102.40	102.40	102.40
Si	C	Si	109.47	117.55	106.03	106.03	109.19
Si	C	Sn	109.47	110.39	109.19	109.19	107.60
Si	C	Sn	109.47	121.83	124.94	124.94	126.76
Si	C	Sn	109.47	109.35	103.03	103.03	100.56

Figure 9: Top: Target crystal, $SnCSi_3$, for the run discussed in this section. Purple atoms are Sn, yellow ones are Si, and black is C. Below it, the crystal structures associated with the maximum similarity of the given stage are shown. The crystal structures are illustrated with the program XCrySDen [31]. For this visualization, the bond lengths of the structures are adjusted to the covalent radii, which is set to 1 Å for C, while for Si and Sn the standard covalent radii of 1.17 Å and 1.48 Å respectively are used. The two tables at the bottom list the bond lengths and angles between the central C atom and the surrounding Si and Sn atoms in the structure.

The mean similarity (blue line) raises only slightly and quickly reaches a plateau, while the minimal similarity always stays close to 0. This leads to a steady increase of the standard deviation (yellow line). The standard deviation of the volume is already high at the start and stays mostly constant. According to these results, the population can be considered diverse. Looking at the maximum volume (red line), it can be seen that for a short period, the volume of some structures exceeds the maximum volume of 512 Å^3. This is because cell bounds are only enforced for mutations that explicitly change the unit cell. However, some mutations, like the conventional cell mutation (see Table 4), can also change the cell size, but cell bounds are not checked. The crystal structure with the maximum similarity of this stage, is displayed in Fig. 9. It has a similarity of 0.904 to the target structure, its volume of 497.97 Å^3 is by 2.03 Å^3 smaller than the target. Since the found structure has the already same composition as the target, the bond distances and angles can be compared to verify their similarity. The mean difference between the distances is 0.214 Å, for the angles it is 11.73°[4]. The structure of stage 1 can be considered similar to the target, since it has the correct composition and a similar arrangement of the atoms. After 115 generations, the break condition was reached due to the similarity fitness reaching a value higher than 0.90.

Stage 2: Optimization

All individuals of the first stage are used as the initial population of the second stage. This stage focuses on optimizing their similarity fitness. Only rattle, permutation, and the conventional cell mutation, as well as the one point element crossover are used (see Tab. 3), to ensure that no new structures are created but only the existing ones are optimized. These mutations are relatively cheap to compute and therefore allow for a fast optimization process. The *number of atoms fitness function* is not used in this stage to avoid prioritizing small structures with low similarity. As seen in Fig. 8, the standard deviation of the fitness and the volume decreases, indicating that the population is getting less diverse. The maximum similarity reached in this stage is 0.992, the corresponding crystal structure is shown in Fig. 9. Its unit cell volume is 498.28 Å^3, i.e., only 1.72 Å^3 smaller than the targets' cell volume. The cell size is not limited, since the mutations and crossovers used cannot drastically increase the size anyway. The mean difference in bond length, compared to the target is 0.105 Å, that of the bond angles 5.536°. Therefore, with respect to both parameters, the structure found at this stage, is more similar to the target than the one found in the first stage. The break condition was met after 24 generations due to the maximum similarity of $S_{\text{max}} = 0.99$ being reached.

Stage 3: Extended exploration

The final population of the second stage is used as the start population for the third stage. Now larger unit cell volumes are allowed, and the parameter governing the maximum number of atoms of the *number of atoms fitness function* is also increased.

[4]The mean differences are calculated with the arithmetic mean of the absolute differences. The correct order of the atoms is not important, since $SnCSi_3$ has the same bond lengths and angles for all combinations of the atoms.

The volume limit defined by these cell bounds is 2744 Å^3. Mutations and crossovers are the same as in stage one. The drastic increase in the standard deviation of the fitness and volume indicate that the population is becoming diverse again. The crystal with the maximum similarity of $\mathrm{S_{max}} = 0.994$ is displayed in 9. While its maximum similarity is higher than in stage 3, the volume, bond angles and lengths of the best found structures are the same. No clear trend is visible in these results, since the break condition was reached after only five generations. The low number of stages is due to the fact that the structure found in stage 2 is already similar enough to the target and therefore no further optimization was needed.

Stage 4: Final optimization

The previous population was carried over to the fourth stage. This stage is very similar to the second, but only a single similarity fitness (see Table 2) with a gamma of 0.1 is used. This is done to reduce the diversity of the population, to focus only on the best structures found in the previous stages. The break condition of this stage was also reached after five generations. Similar to the second stage, the standard deviation of the fitness and the volume decreases, but no clear trend can be seen due to the small number of generations. The maximum fitness reached in this stage is 0.996. The corresponding crystal can be seen in Fig. 9. The mean difference in bond distances is 0.102 Å, the mean difference in bond angles is 5.99°. While the former are better than in the previous stage, the latter are slightly worse.

Testing the robustness of the GA

Many aspects of the GA rely on a random number generator, which uses a seed to create a deterministic sequence of semi-random numbers. Different seeds should not lead to drastically different results, since the success of the GA should not depend on its starting conditions. To test the randomness stability of the GA, the configuration from the previous section was run with the 10 different seeds. For each run, the maximum similarity $\mathrm{S_{max}}$ and the total number of generations $\mathrm{N_g}$ performed until the run was stopped by its break condition, are shown in Tab. 5. Each run is associated with the displayed ID for easier identification. All similarity values are rounded to the third decimal place in order to visualize the differences between them. The mean value for $\mathrm{S_{max}}$ is 0.983, with a standard deviation of 0.022 and a range of 0.071. While the low standard deviation indicates that the GA produces consistent results, the noticeably reduced maximal similarity of $S_{max} = 0.925$ in run 9 shows that the GA can produce non-optimal results. The best result is achieved

ID	1	2	3	4	5	6	7	8	9	10
$\mathrm{S_{max}}$	0.962	0.991	0.994	0.996	0.996	0.991	0.986	0.996	0.925	0.996
$\mathrm{N_g}$	271	291	98	153	275	143	350	104	349	161

Table 5: Results of the robustness test of the GA configuration. $\mathrm{N_g}$ is the number of generations needed, to reach the break condition.

in the runs 4, 5, 8, and 10, which all have a maximum similarity of $S_{max} = 0.996$. Additionally, the number of generations $\mathrm{N_g}$ needed, differs between the runs. It is especially high in the worst performing run, where the GA needed 349 generations to perform the optimization. In the run with the lowest $\mathrm{N_g}$, run 3, it needed only 98 generations and still achieved the highest similarity of $S_{max} = 0.996$.

5.3 General applicability of the GA workflow

To test if the algorithm is able to achieve a more general inversion, meaning that it is not limited to single structure, it was tested on the descriptors of 150 different target structures. From each of the three different datasets (Ternary, MP, Kaggle-NOMAD) discussed in Section 4.3 50 crystals are randomly selected as the target, to cover a wide range of different crystal. As a compromise between run time and similarity, a run is considered successful if it has a maximum similarity of $\mathrm{S_{max}} \geq 0.9$. The break conditions for each stage are adjusted to stop the GA if it reaches a maximum similarity of 0.9. All other settings of the GA are the same as in the previous section. The calculations of each database were run in parallel on the group's computer cluster Dune, with a maximum runtime of 4 days, utilizing 36 CPU cores each.

Table 6 gives an overview of some important results of this generalization test. The presented data are separated into finished (f) runs and unfinished (uf) ones, which were stopped upon the maximum runtime being reached. Since the GA saves data only after the completion of each stage, the unfinished runs might miss information about the last stage that was not fully completed. Only $\mathrm{N_{f,\,total}} = 47$ runs were finished in time, which translates to 31% of all runs. Of all runs on the MP and Ternary datasets, 52% and 42%, respectively, were completed, while no runs finished on the Kaggle-NOMAD dataset. In total $\mathrm{N_{S\geq 0.9}} = 21$ runs can be considered successful, which is 14% of all runs and 45% of the finished ones. The similarity standard deviation is $\sigma(\mathrm{S_f}) = 0.31$ for the MP and Ternary datasets.

Database	$\mathrm{N_f}$	$\mathrm{N_{S>0.9,f}}$	$\overline{\mathrm{S}}_\mathrm{f}$	$\sigma(\mathrm{S_f})$	$\mathrm{N_{comp.,\,f}}$	$\mathrm{N_{stoich.,\,f}}$	$\mathrm{N_{stoich.}}$
mpData	26	11	0.74	0.31	5	26	48
ternary	21	10	0.69	0.31	10	19	36
kaggle	0	0	-	-	0	0	12

Table 6: Overview of results that were obtained by running 150 GAs with different target structures, as discussed in this section. $\mathrm{N_f}$ is the number of finished runs, $\mathrm{N_{S>0.9,f}}$ is the number of finished runs that ended with a maximum similarity of 0.9 or higher. The average maximum similarity for all finished runs is denoted by $\overline{\mathrm{S}}_\mathrm{f}$ and its standard deviation is $\sigma(\mathrm{S_f})$. Additionally, $\mathrm{N_{comp.,\,f}}$ is the number of finished runs in which the correct composition was found and $\mathrm{N_{stoich.,\,f}}$ is the number of finished runs in which the correct stoichiometry was found. And lastly, $\mathrm{N_{stoich.}}$ is the total number of runs in which the correct stoichiometry could be found.

Overall, in nearly all finished and even unfinished runs, the found structures have the correct stoichiometry. This is true for the MP dataset, while for the Kaggle-NOMAD dataset only 12 runs were able to find the correct stoichiometry. Figure 10 presents two histograms illustrating the distribution of the maximum similarity of all runs on each database. The left histogram consists of all finished runs and the right one of all unfinished runs. The x-axis is divided into bins of width 0.1, with the corresponding upper bounds being displayed as the x-labels. Looking at the left histogram, it can be seen that runs either end up with a similarity close to 0.0 or 1, and only a few are in between.

To further analyze the results, some relations in the collected data are shown in Fig. 11. Finished runs are shown in the top, unfinished runs in the bottom plots. The data points of each dataset are marked with different colors. The plots in the middle shows how the target atomic density $\rho_{\mathrm{target}} = \mathrm{N_A}/\mathrm{V}$ is related to the atomic density of the most similar structure ρ_{best} found in each run. A clear correlation is visible, since the data points are close to the diagonal, which indicates that runs found structures with a very similar atomic density to the target. Looking at the unfinished runs, this correlation is also present, even though not all stages are completed. The relation between the maximum similarity $\mathrm{S_{max}}$ and the density of the target structure is shown in the left plot. Only runs where the target densities is below 0.07 atoms/$\mathrm{Å}^3$ did finish. One can also see that all selected structures from the Kaggle-NOMAD dataset are above this threshold. For densities around 0.05 atoms/$\mathrm{Å}^3$, both finished and unfinished runs are able to find structures with a wide range of similarities, so no clear correlation is evident. However, target structures with very low densities are found reliably. The data for the uncompleted runs show a decrease in similarity with increasing density, but since the data are incomplete, no clear statement can be made.

The plot on the left shows the relation between $\mathrm{S_{max}}$ and the number of generations each run needed to finish. Looking at the completed runs, it can be seen that the found similarity decreases with increasing number of generations, and nearly all successful runs finished in the first 200 generations. Additionally, all runs that reached a similarity lower than approximately 0.7 end after 300 generation. Looking at the unfinished runs, they are mostly finished around 100, 150, and 250 generations, meaning that most of their stages were also stopped due to insufficient similarity. One can assume that if the GA is not successful in the first 200 generations, it will most likely not be successful at all. This heuristic could be used to abort the unsuccessful runs even earlier to save computational resources. However, additional verification is necessary, using more runs and assuring that no bias is present that occurs due to the limited runtime of the computations.

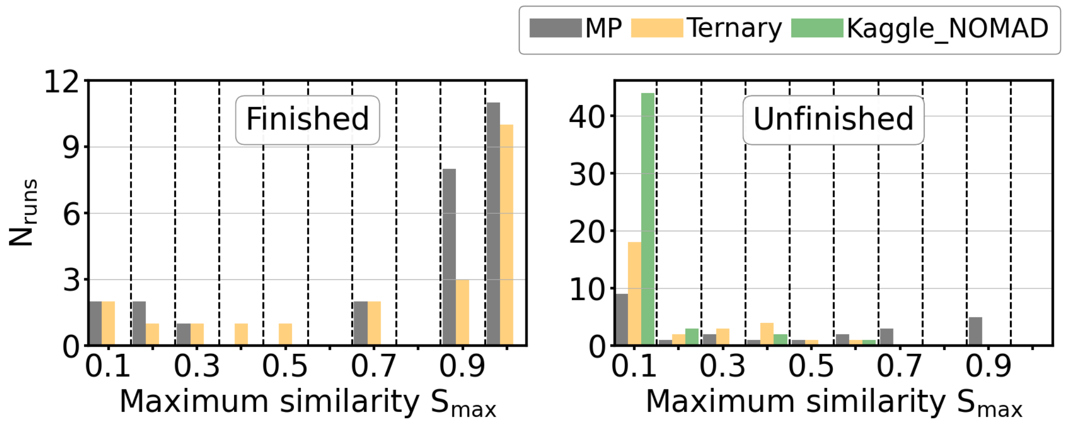


Figure 10: Histogram of the highest similarities of the 150 runs.

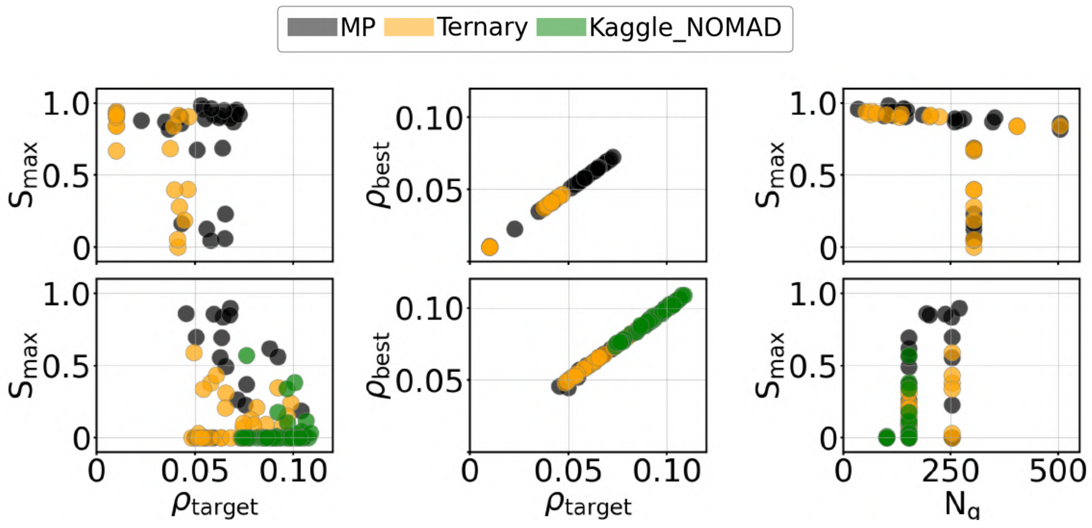


Figure 11: Generalization test results for the GA. The top row shows all data points of the finished runs and the bottom row of the unfinished runs. The plots in the left column show the best similarity found by the GA for each target density. The middle plots show the best similarity found by the GA for each target density compared to the target density. The right plots show the best similarity found by the GA for each target density compared to the number of generations.

5.4 Customizing the GA for complex target structures

While the previous results show how the default FUCrIMODo workflow can find different target structures, the following results present how the workflow can be extended to find a specific target.

The crystal structure of the clathrate $Al_{16}Ba_8Si_{30}$ (see Section 4.3) was selected as the target due to its complex arrangement of atoms, which makes it a good candidate to test the capabilities of the methods presented in this work. The task was approached by repeatedly running the GA with different configurations and using the analysis tools provided by FUCrIMODo. Based on the findings of the analysis, adjustments to the GA configuration were made to improve its performance. Additionally, the found structures were fed back into a new GA run to further optimize them. Since the final workflow is very complex, only a few key aspects and concepts of its implementation will be discussed. The workflow consists of multiple *phases*, where each phase consists of multiple stages that are designed to achieve a common goal.

Build up

The idea of the build-up phase is to first create structures that have a descriptor with high similarity to a specific part of the target's SOAP descriptor. Then these individuals are combined to form structures with descriptors that are similar to the whole target SOAP. Two types of stages are used for this, the *partial exploration* and the *recombination* stages. The partial exploration stages use genetic operators similar to the ones used in the exploration stages explained in Section 5.2. Each partial exploration stage introduces a set of 16 new single-atom structures to the population and optimizes the combined population with a set of species-specific fitness functions (see Table 2). This leads to a very diverse population consisting of structures that are optimized for individual parts of the SOAP. After each partial exploration stage, a recombination stage is applied. A recombination stage has a high crossover probability of 0.8 and uses the similarity fitness function of the whole SOAP, with $\gamma = 0.001$. This enables the GA to recombine the partially optimized structures to form new individuals that are similar to the whole target SOAP. Since the target has three different species, Al, Ba and Si, six different species specific fitness functions are used, namely Al-Al, Al-Ba, Al-Si, Ba-Ba, Ba-Si, and Si-Si (compare also Fig. 2, showing individual parts of the SOAP). The initial population of the GA consists of 16 randomly generated single-atom structures. Then the first stage is applied, which is a partial exploration stage that introduces 16 new single-atom structures to the population and optimizes the structures for Al-Al interactions. After that, a recombination stage is performed that uses the similarity fitness function of the whole SOAP with $\gamma = 0.001$ and also the species-specific fitness function for Al-Al interactions. The third stage adds additional 16 single-atom structures to the population and optimizes the structures for Al-Al and Al-Ba interactions. Since new single-atom structures are added to the population, the GA can explore new structures and is not limited to the ones that were found in the previous stages. The fourth stage is similar to the second and reintroduces the similarity fitness function of the whole SOAP. This process is repeated four times, where each time one of the remaining species-specific fitness functions is added. In total, 12 stages are performed, resulting in an $Al_8Ba_4Si_{15}$ structure with a reference similarity S_r of 0.089. Since the similarity is still very low, the population is passed on to the next phase.

Exploration and optimization

The exploration and optimization phases work the same way as the GA configuration explained in Section 5.2, meaning that exploration and optimization stages alternate. The first stages allow for unit cells with lattice constants up to 11 Å, and after four stages, the maximum lattice constant is increased to 14 Å. In total, 11 stages were performed during this phase and found an $Al_{16}Ba_8Si_{30}$ structure with a reference similarity of 0.827. This can already be considered a good result, since the found structure has the correct composition, but to reach a higher similarity, an additional optimization phase is needed.

Final optimization

For the final optimization phase, the algorithm is run with many different configurations of the GA stages. The idea is to create a configuration and run it for a few stages to test its performance. It is then adjusted and executed again on the individuals created in the previous run. With this approach, different GA configurations can be tested while simultaneously optimizing the structures. To reduce the bias of knowing the correct target structure, mutations and crossovers that can change the composition are still used during the early runs of the final optimization even though the correct composition was already found. But as the GA progresses, analysis of the similarity evolution shows that only the rattle mutation (see Table 4) is able to further increase the similarity. In total, 21 runs with different configurations were performed. After the fourth run, a maximum similarity of 0.90 is obtained, but in the following runs the rate at which the similarity increases drastically decreases. After reaching a maximum similarity of 0.95, the GA is not able to significantly increase this value anymore and is therefore stopped. These results are discussed in the following.

Analysis of the optimization process

Since the configurations of the runs have vastly different parameters, like the number of generations, the population size, or the used operators, and were often run in parallel with other test configurations, no clear statement can be made about the total runtime of the search process. Including the whole process of analysis and adjustment of each run, it took a few weeks to reach a similarity of 0.95. Figure 12 shows the corresponding structure (right) together with the target structure (left). It can be seen in both structures that Si and Sn atoms form pentagonal arrangement and encapsulate Ba atoms in between them. But a sufficient visual analysis is not possible due to the large number atoms and the complexity of the structure. Therefore, other descriptors must be compared to verify the similarity of the structures. First, the volume of the found structure is 1162.58 Å^3 which is only 0.05 Å^3 bigger than the target structure that has a volume of 1162.53 Å^3. As seen in Fig. 12, the cell shape differs between the structures. For further analysis, the RDF (see Section 4.1) of the target structure (shown in blue) is compared to that of the found structure (in orange) in Fig. 13. The first three peaks of the RDFs of both structures are at a similar position, but for the found structure the peaks are broader and less high. This can be explained by the fact that the atomic positions in the found structure are slightly off due to the statistical nature of the GA. For larger distances, some peaks, like the one at around 9 Å, are present in both structures, but again, those for the found structure are broader and less high. Overall, the comparison between the RDFs indicates that the structures exhibit indeed similarities, and for the scope of this work, the reconstruction can be considered successful.

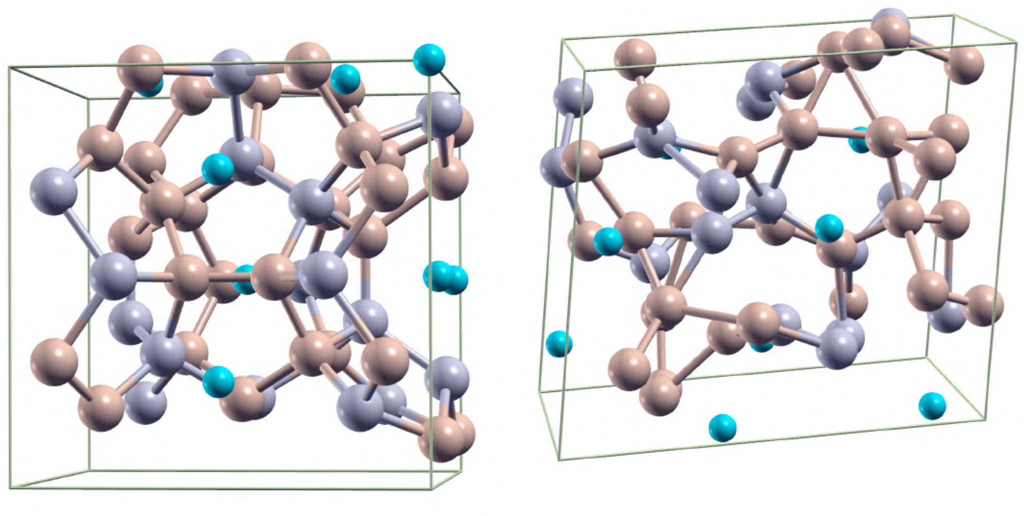


Figure 12: Comparison of the target structure (left) and the structure found by the algorithm (right). Blue atoms are Ba, silver atoms are Al and yellow ones are Si. The structures are visualized with XCrySDen [31], and bond lengths are adjusted to the covalent radii, set to 1.5 Å for Al and Si, as well as 1.0 Å for Ba.

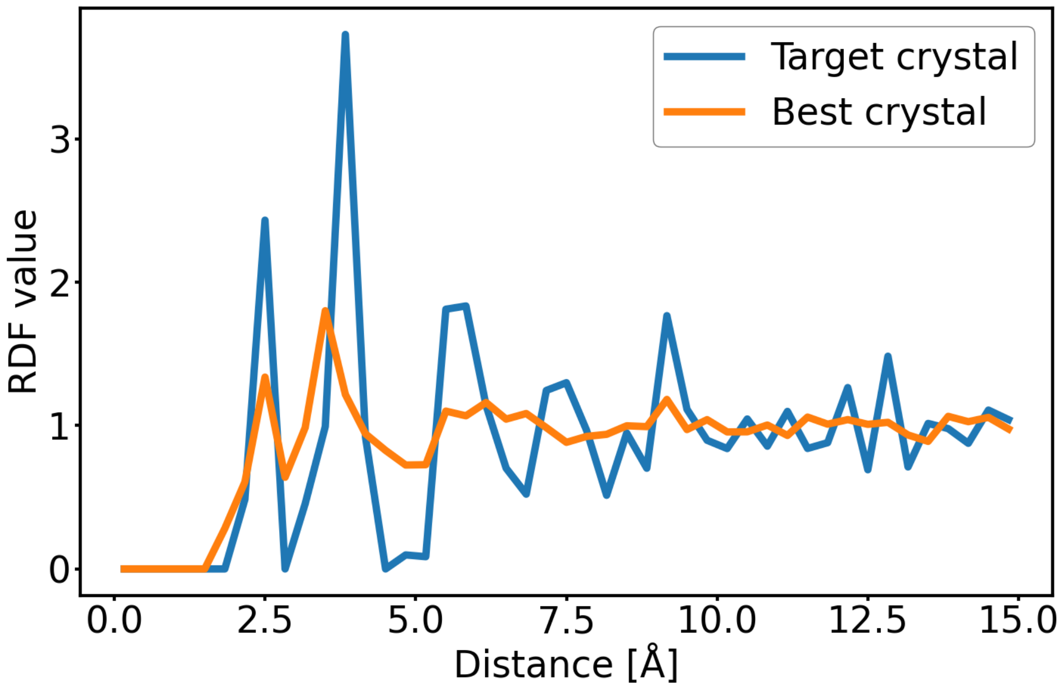


Figure 13: Comparison of the RDFs of the structures shown in Fig. 12. The parameters used to calculate the RDF are $r_{\text{max}} = 15$ Åand $N_{\text{bins}} = 45$. Since the structures have a size smaller than 15 Å, supercells were used to calculate the RDF.

6 Discussion

This work addresses the problem that widely used crystal structure representations like the SOAP descriptor cannot be used in generative ML models, since they are currently not invertible. While other approaches to solve this issue include the creation of new representations that are specifically designed to be invertible[10], this work focuses on the creation of a statistic inversion algorithm for the widely used SOAP descriptor. In the literature, two methods have been reported that are also able to invert this descriptor, but they are limited to the local SOAP descriptor and can therefore only be used in cases where the target composition is known already [2, 4]. This work, on the other hand, proofs that a novel multi-stage GA is able to invert global SOAP descriptors without the need for any knowledge about the target structure other than its SOAP descriptor. This makes its potential use in generative models much more versatile. While the results of this work are promising, there are still some limitations that will be discussed in the following.

The first result, described in Section 5.2, is the inversion of the descriptor of a simple target structure. To confirm that the found structure is indeed similar to the target, their bond angles and lengths were compared (see Fig. 9). Some of them do not match perfectly, but overall the structures are very similar. While this result could be improved by increasing the threshold for the break condition to a higher maximum similarity, this would also increase the runtime of the GA, which was beyond the scope of this thesis. Using operators that structure the atoms in the unit cell, thereby taking into account the atomic forces, may be a better solution for finding the correct atomic positions. For example, Ref. [10] also uses such a structure optimization step in their inversion process. Testing the robustness of the GA revealed that the number of generations required to find the target structure, can vary. This is likely due to the fact that the parameters of the GA were chosen heuristically and not through a systematic process, leading potentially to a sub optimal GA configuration.

Section 5.3 presented a large-scale test of the GA to show its general applicability. The results demonstrate that the used configuration of the GA is not able to reach this threshold for all target structures. When comparing the density of the target structure and the reference similarity, no clear correlation could be found (see Fig. 11). While not shown in this thesis, the number of atoms and volume of the target were also compared to the success rate, but did also not show a correlation. Therefore, no clear statement can be made for the reasons why some target structures could not be recovered. Furthermore, the results show that the algorithm has limitations in runtime for target structures with an atomic density of 0.7 or higher. This may be explained with inefficiencies in the GA implementation. Most of the used mutations and crossovers work by randomly applying modifications to the structures until a valid offspring is found (see Section 5.1). For example, the *add atoms* mutation (see Table 4) randomly places atoms in the given unit cell which can lead to a high number of retries if the structure is already densely packed. Later analysis revealed that some runs could not be finished due to inefficient parallelization. This is holds especially true for the dense structures in the Kaggle-NOMAND dataset.

In Section 5.4, the target structure, $Al_{16}Ba_8Si_{30}$, was chosen due to its complexity and the fact that it was necessary to create a manual GA configuration to retrieve it from its SOAP descriptor. The presented GA configuration is not optimized to be as efficient as possible but to show how the process of manually optimizing the GA can look like. During this process, it was discovered that some mutation were not correctly configured to optimize the structures. They modified the structures so much, that instead of improving the ones that were already found, they create entirely new structures. Such insights are very valuable, since they will lead to more efficient GA configurations in future applications. While not further discussed here, a run was already tested with an improved GA configuration and could perform the inversion in a fraction of the time. In the end, the GA found a structure that has a reference similarity of 0.95, which is a good result especially considering the complexity of the target structure. The comparison of the RDFs of the target and found structure (see Fig.13), especially the differences in peak widths underlines the need for methods to symmetrize the atomic positions. Adding such a method, would most likely improve the maximum similarity beyond 0.95. Finally, the premise of the whole process was that no information about the target structure other than the SOAP descriptor should be used during the creation of new configurations. While this restriction was strictly followed, it cannot be guaranteed that conclusions made from analyzing the created structures are not biased by the knowledge of the target structure.

Section 5.1 described the program FUCrIMoDo that was created to implement the novel multi-stage GA. All aspects of FUCrIMoDo are built in a way that allows for easy extension and modification of the existing code. While some testing of the program has been done to ensure that the results can be trusted, a much higher test coverage would be needed to ensure that the program fully works as intended. Another aspect that needs to be improved is the feedback that the program gives regarding the success rate of mutations and crossovers. While the current logging system provides a lot of information about all operations, no analysis tools are implemented to efficiently determine what mutations and crossover are responsible for the success of a run. This is crucial for optimizing the GA for future applications.

7 Conclusions and Outlook

This work presents a method that is capable of constructing crystal structures based solely on their SOAP descriptor. Even large and complex structures like $Al_{16}Ba_8Si_{30}$ (see Section 5.4) could be successfully reconstructed, which demonstrates the potential of the method. Using the novel multi-stage GA, a vast space of possible crystal structures can be efficiently sampled to find the structure that is most similar to the target. The program FUCrIMODO provides the necessary framework to create such multi-stage GA workflows due to its modular design and easy configuration. While the current state of the program is not able to reconstruct all the tested target structures, it can already reliably find their stoichiometry and atomic density, which is a very good starting point for further improvements. Furthermore, the possibility of a streamlined analysis of the results with Jupyter notebooks (see Fig. 7) and other tools allows for the effective implementation of improved workflows.

The presented methods promise to be usable in the creation of a fully automated general inversion process for the SOAP descriptor, enabling its use in generative models. But to realize the full potential of the method, several steps must be taken. First, the implementation of the program itself must be thoroughly tested to ensure that it is working correctly under all circumstances. Next, the program must be optimized for efficiency and speed so that more structures can be tested to find the limits of the method and improve it. Furthermore, an automated workflow must be created that can find the best possible configuration of the program. This can be done, for example, with genetic programming, which is a subset of evolutionary algorithms that applies principles similar to GAs to the program's workflow [21]. In fact, the stage design was intended to be used in this way from the beginning. Lastly, to fully test the capabilities of this method, a generative model should be created that can predict the SOAP descriptor of a structure based on desired properties. If the GA can then reconstruct structures from the model's predictions and their properties can be verified, a powerful tool for material discovery would be created.

8 References

References

[1] Sergey N. Pozdnyakov, Michael J. Willatt, Albert P. Bartók, Christoph Ortner, Gábor Csányi, and Michele Ceriotti. On the Completeness of Atomic Structure Representations. *Physical Review Letters*, 125(16):166001, October 2020.

[2] Matteo Cobelli, Paddy Cahalane, and Stefano Sanvito. Local inversion of the chemical environment representations. *Physical Review B*, 106(3):035402, July 2022.

[3] Piyush M. Tagade, Shashishekar P. Adiga, Shanthi Pandian, Min Sik Park, Krishnan S. Hariharan, and Subramanya Mayya Kolake. Attribute driven inverse materials design using deep learning Bayesian framework. *npj Computational Materials*, 5(1):127, December 2019.

[4] Albert P. Bartók, Risi Kondor, and Gábor Csányi. On representing chemical environments. *Physical Review B*, 87(18):184115, May 2013.

[5] Condensed-matter physics | Solid State, Quantum Mechanics, Superconductivity | Britannica. https://www.britannica.com/science/condensed-matter-physics.

[6] Lauri Himanen, Marc O.J. Jäger, Eiaki V. Morooka, Filippo Federici Canova, Yashasvi S. Ranawat, David Z. Gao, Patrick Rinke, and Adam S. Foster. DScribe: Library of descriptors for machine learning in materials science. *Computer Physics Communications*, 247:106949, February 2020.

[7] Bing Huang, Guido Falk Von Rudorff, and O. Anatole Von Lilienfeld. The central role of density functional theory in the AI age. *Science*, 381(6654):170–175, July 2023.

[8] Christopher Sutton, Luca M. Ghiringhelli, Takenori Yamamoto, Yury Lysogorskiy, Lars Blumenthal, Thomas Hammerschmidt, Jacek Golebiowski, Xiangyue Liu, Angelo Ziletti, and Matthias Scheffler. NOMAD 2018 Kaggle Competition: Solving Materials Science Challenges Through Crowd Sourcing. *Kaggle*, 2018.

[9] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. *ACS Central Science*, 4(2):268–276, February 2018.

[10] Hang Xiao, Rong Li, Xiaoyang Shi, Yan Chen, Liangliang Zhu, Xi Chen, and Lei Wang. An invertible, invariant crystal representation for inverse design of solid-state materials using generative deep learning. *Nature Communications*, 14(1):7027, November 2023.

[11] Trent Barnard, Steven Tseng, James P. Darby, Albert P. Bartók, Anders Broo, and Gabriele C. Sosso. Leveraging genetic algorithms to maximise the predictive capabilities of the SOAP descriptor. *Molecular Systems Design & Engineering*, 8(3):300–315, 2023.

[12] Henning Heider and Thorsten Drabe. A cascaded genetic algorithm for improving fuzzy-system design. *International Journal of Approximate Reasoning*, 17(4):351–368, November 1997.

[13] David E. Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3, 1989.

[14] R. J. D. Tilley. *Crystals and Crystal Structures.* Wiley, Hoboken, NJ, second edition edition, 2020.

[15] Ask Hjorth Larsen, Jens Jørgen Mortensen, Jakob Blomqvist, Ivano E. Castelli, Rune Christensen, Marcin Dułak, Jesper Friis, Michael N. Groves, Bjørk Hammer, Cory Hargus, Eric D. Hermes, Paul C. Jennings, Peter Bjerre Jensen, James Kermode, John R. Kitchin, Esben Leonhard Kolsbjerg, Joseph Kubal, Kristen Kaasbjerg, Steen Lysgaard, Jón Bergmann Maronsson, Tristan Maxson, Thomas Olsen, Lars Pastewka, Andrew Peterson, Carsten Rostgaard, Jakob Schiøtz, Ole Schütt, Mikkel Strange, Kristian S. Thygesen, Tejs Vegge, Lasse Vilhelmsen, Michael Walter, Zhenhua Zeng, and Karsten W. Jacobsen. The atomic simulation environment—a Python library for working with atoms. *Journal of Physics: Condensed Matter*, 29(27):273002, 2017.

[16] H. Föll. Bravais Lattices and Crystals. https://www.tf.uni-kiel.de/matwis/amat/iss/kap_4/illustr/s4_2_1.html.

[17] Jarno Laakso, Lauri Himanen, Henrietta Homm, Eiaki V Morooka, Marc OJ Jäger, Milica Todorović, and Patrick Rinke. Updates to the DScribe library: New descriptors and derivatives. *The Journal of Chemical Physics*, 158(23), 2023.

[18] Benedikt Hoock. NOMAD dataset: Descriptors_ternaries_data_exciting, 2022.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[20] E. L. Willighagen, R. Wehrens, P. Verwer, R. De Gelder, and L. M. C. Buydens. Method for the computational comparison of crystal structures. *Acta Crystallographica Section B Structural Science*, 61(1):29–36, February 2005.

[21] David B. Fogel, Thomas Bäck, and Zbigniew Michalewicz, editors. *Evolutionary Computation.* Institute of Physics Publishing, Bristol ; Philadelphia, 2000.

[22] Wojciech Paszkowicz. Genetic Algorithms, a Nature-Inspired Tool: A Survey of Applications in Materials Science and Related Fields: Part II. Paper, Institute of Physics, Polish Academy of Sciences, Warsaw, Poland, 2013.

[23] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagn\' e. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13:2171–2175, July 2012.

[24] Sameer Agarwal Kalyanmoy Deb, Amrit Pratap and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), 2002.

[25] Lauri Himanen, Patrick Rinke, and Adam Stuart Foster. Materials structure genealogy and high-throughput topological classification of surfaces and 2D materials. *npj Computational Materials*, 4(52), 2018.

[26] Benedikt Hoock, Santiago Rigamonti, and Claudia Draxl. Advancing descriptor search in materials science: Feature engineering and selection strategies. *New Journal of Physics*, 24(11):113049, November 2022.

[27] Tim Bechtel, Daniel T. Speckhard, Jonathan Godwin, and Claudia Draxl. Band-gap regression with architecture-optimized message-passing neural networks, September 2023.

[28] Maria Troppenz, Santiago Rigamonti, and Claudia Draxl. Predicting Ground-State Configurations and Electronic Properties of the Thermoelectric Clathrates $Ba_8Al_xSi_{46-x}$ and $Sr_8Al_xSi_{46-x}$. *Chemistry of Materials*, 29(6):2414–2424, March 2017.

[29] Python Software Foundation. ABC: Abstract Base Classes. Python Software Foundation, 2024.

[30] Executable Books Community. Jupyter Book. Zenodo, February 2020.

[31] Anton Kokalj. XCrySDen—a new program for displaying crystalline structures and electron densities. *Journal of Molecular Graphics and Modelling*, 17(3-4):176–179, June 1999.

A Appendix for Section 5

A.1 GA configuration and parameters for Sec. 5.2

Atom 1	C			Si		Sn
Atom 2	C	Si	Sn	Si	Sn	Sn
Dist. [Å]	1.2	1.5	1.7	1.8	2.0	2.2

Table 7: Table with the minimum allowed distances for atoms of two species. They are calculated between different species where calculated by the `closest_distance_generator` of the ASE library, with a ratio of covalent radii of 0.8. The values are rounded to the first decimal.

	a, b, c [Å]	α, β, γ [°]	ϕ, χ, ψ [°]
Bounds 1	[1, 8]	[20, 160]	[0, 180]
Bounds 2	[1, 14]	[20, 160]	[0, 180]

Table 8: Bounds of the cell used in the run discussed in the section. The parameters are explained in Section 3.

Stage	Condition
1	$(f_{max} \geq 0.90 \wedge N_{\mathrm{G}} \geq 5) \vee (f_{max} \leq 0.70 \wedge N_{\mathrm{G}} = 100) \vee (N_{\mathrm{G}} = 200)$
2	$(f_{max} \geq 0.99 \wedge N_{\mathrm{G}} \geq 5) \vee (f_{max} \leq 0.90 \wedge N_{\mathrm{G}} = 50) \vee (N_{\mathrm{G}} = 100)$
3	$(f_{max} \geq 0.90 \wedge N_{\mathrm{G}} \geq 5) \vee (f_{max} \leq 0.70 \wedge N_{\mathrm{G}} = 100) \vee (N_{\mathrm{G}} = 200)$
2	$(f_{max} \geq 0.99 \wedge N_{\mathrm{G}} \geq 5) \vee (N_{\mathrm{G}} = 100)$

Table 9: Break conditions used in the section. The value f_{max} is the maximum similarity fitness value for $\gamma = 0.1$ and N_{G} is the number of generations of the current stage.

Fitness

Name	Parameters	Stages
Physicality	closest dist.=see Tab. 7	all
Number of atoms	max. atoms=[10,20]	1,3
SOAP similarity	SOAP params.=see Tab. 1	all
	kernel=rbf, with γ=(0.1, 0.001)	
Species similarity	kernel=rbf, with γ=0.1; weight=1/3	all
	comp. species=(C, Ge-Si, Ge, C-Si, Si, C-Ge)	
	species=(C, C, Ge, Ge, Si, Si)	

Crossovers (probability = 0.8)

One point element	force changes=False	all
Unit cell	scale atoms=True; cell bounds= [Bound 1, Bounds 2]	1,3
Stack cells	scale atoms=True; cell bounds= [Bound 1, Bounds 2]	1,3

Mutations (probability = 0.8)

Rotation		1,3
Soft		1,2,4
Min. tilt		1,2,4
Conv. cell	tolerance = 0.3	all
Delete	number of atoms=1	1,3
Enlarge	cell bounds = [Bound 1, Bounds 2]	1,3
Strain	cell bounds = [Bound 1, Bounds 2]	1,3
Permutation	probability=0.5; n top=all	all
Rattle	probability=0.5; n top=all; strength=0.8	all
Replace	allowed species = C, Ge, Si; N atoms = 1	1,3
Cutout	tolerance = 1.; cell bounds = [Bound 1, Bounds 2]	1,3
Multi	mutations=all described mutations, chain n mutations = 2	1,3
Add	allowed species=C, Ge, Si; N atoms=1	1,3
	only same species=False	

Table 10: Table of fitness, crossover and mutation classes that where used in the run the described in section 5.2. The stage column shows in which stages the operators where used. Parameters in round brackets (...) show that the module was defined multiple times with different parameters. Parameters in square brackets [...] show the different values that where used in the different stages. All parameters that are not mentioned in the table where set to their default values. All mutations and crossovers used the closest distances from table 7 to ensure physicality. The cell bounds *Bounds 1* and *Bounds 2* are defined in Tab. 8.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen, Bilder sowie die Nutzung von Künstlicher Intelligenz für die Erstellung von Texten und Abbildungen, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, [illegible] ....