**StructOpt User’s Guide**

A general atomic structure optimization program

University of Wisconsin Computational Materials Group

StructOpt User’s Guide (v. 1.0)

Developed by the CMG at the University of Wisconsin-Madison under Dr. Dane Morgan

Table of Contents

Introduction 3

Installation 3

Basic Installation 3

Parallel Installation 4

Quick Start 5

Algorithm Overview 8

Random Search 8

Basin Hopping 8

Simulated Annealing 9

Genetic Algorithm 10

Parallelization Schemes 11

Input and Output Syntax 13

Input Options 13

Output Files 21

Post-Processing 24

Functionalities 25

Primary Parameters 25

Output Parameters 27

Post-Processing Parameters 28

General Algorithm Parameters 29

Population and Individual Generation Parameters 33

Evaluation Parameters 37

Crossover Parameters 40

Mutation Parameters 41

Selection Parameters 47

Reference Listing 54

Case Studies 60

User Expansion 61

Individual Class Object 61

Optimizer Class Object 62

Basic Algorithm Flow 62

Seeding a Population 64

Adding New Fitness Functions 64

Adding New Mutations or Moves 66

Adding New Crossover Functions 66

Expansions with MAST 67

Error Messages 70

General Warnings 70

LAMMPS Execution error 70

Large Energy Warning 71

# Introduction

This document serves as a user’s guide for StructOpt. StructOpt is a general optimizer for python targeted at identifying stable atomic structures. StructOpt was designed to provide both flexibility and simplicity. While there are many optional parameters to allow the user great customizability, there are also many default settings that allow for faster more simple runs. The [Quick Start](#_Quick_Start) section of this document provides details on the basic information required to perform an optimization. Additional details on input and output structures can be found in the [Input and Output Syntax](#_Input_and_Output) section of this document. Details on the many options currently available for StructOpt are provided in the [Functionalities](#_Functionalities) section of this document. A list of these options is also provided in the [Reference Listing](#_Reference_Listing) section of this document. The [Case Studies](#_Case_Studies) section provides some example problems and serves as a general tutorial. Useful tips for expanding StructOpt with additional user subroutines or options can be found in the [User Expansion](#_User_Expansion) section of this document. Finally, an explanation of commonly generated errors and troubleshooting advice is provided in the section entitled [Error Messages](#_Error_Messages).

# Installation

## Basic Installation

StructOpt is written in python 2.7 and as such requires a working python installation. StructOpt has not currently been tested with alternative versions of python. Additionally, StructOpt has the following python module dependencies:

* Atomic Simulation Environment (ASE) (<https://wiki.fysik.dtu.dk/ase/>)
* Numpy (<http://www.numpy.org/>)
* Scipy (<http://www.scipy.org/>)
* Matplotlib (for visualization) (<http://matplotlib.org/>)
* MPI4PY (for parallelization) ([http://pythonhosted.org//mpi4py/usrman/index.html#](http://pythonhosted.org//mpi4py/usrman/index.html))

It is recommended that StructOpt be used with the Enthought Canopy python distribution (<https://www.enthought.com/products/canopy/>) as this includes the Numpy, Scipy, and Matplotlib modules as well as many other useful modules for python.

Once the required modules are installed, download the zip file from <http://github.com/uw-cmg/MAST> and unzip the file. Then add the location of the StructOpt folder to the python path in your bashrc file.

In order to use StructOpt, an atomic calculation code should also be installed. Currently, StructOpt is equipped for use with LAMMPS (<http://lammps.sandia.gov/>) and VASP (<https://www.vasp.at/>). Please refer to these software packages for installation instructions. The Atomic Simulation Environment (ASE) modules handle communication between StructOpt and these codes. Once one or both of these calculation packages are installed, the user must create an environment variable for the LAMMPS or VASP executable to be accessed by the ASE module. Please see the LAMMPS or VASP page on the ASE website for further information about how to establish these environment variables. Alternatively, for basic testing with StructOpt the native LennardJones calculator in ASE can also be used. See the Quick Start section of this document for more information on running StructOpt.

## Parallel Installation

In order to use the multiprocessor capabilities of StructOpt (this includes both the parallel algorithm and the island method algorithm), the MPI4PY (<http://mpi4py.scipy.org/>) python module and a standard MPI must also be installed. Follow the instructions on the MPI4PY website for information on how to install this module for different MPI versions.

WARNING: For parallel machines that utilize a queuing system or certain versions of MPI, the parallel version of StructOpt may not function. This is because StructOpt will generate many independent processes that may require executing multiple MPI runs from a single MPI connection. Currently, the parallel versions of StructOpt has been tested and proven operational using PBS queuing system and mvapich2 version 1.9.a2. Any suggestions for how to combat this issue are definitely welcome.

StructOpt is distributed as part of the Materials Simulation Toolkit (MAST), which can provide basic parallelization through queue submission without the need to install MPI4PY. However, the user should be aware that the additional overhead of MAST might cause a slow down of the optimization.

# 

# Quick Start

While there are many optional input parameters for StructOpt, the suggested minimum inputs are:

* Type of structure to be optimize
* Type of optimization method
* Number of atoms
* Type of atoms
* File name for bulk solid (dependent on type of simulation)

Currently, users can select from cluster, crystal, or defect in solid (Defect) options for the type of structure to be optimized. Setting the structure type to cluster will engage the defaults for the optimization of a non-periodic group of atoms ideal for simulating nano-clusters. Alternatively, the crystal option allows for the group of atoms specified to be simulated in a periodic environment that allows for alterations to the lattice parameters. Finally, the Defect option allows the specified group of atoms to be added to a bulk solid structure (which must also be provided) for optimization.

The method of optimization can be specified explicitly or customized through a variety of optional input parameters (see the Functionalities section for a description of these customizable parameters). The general optimization methods available in StructOpt include random search, simulated annealing, basin hopping, and genetic algorithm. The default method of optimization is a random search method. Selecting one of the default optimization methods will set a variety of other parameters to specific defaults of those methods. Please refer to the Algorithm Overview section of this document for information on these optimization methods.

Additionally, the number and type of atoms to be simulated must also be provided. This information is provided to StructOpt in a single parameter called atomlist. This parameter is actually a list of tuples that is formatted as follows:

atomlist = [(Type of atom 1, concentration of atom 1, mass of atom 1, chemical potential of atom 1), (Type of atom 2, concentration of atom 2, mass of atom 2, chemical potential of atom 2), (Type of atom 3, concentration of atom 3, mass of atom 3, chemical potential of atom 3), …]

For this parameter, the atom type is provided as a string of the chemical symbol. The concentration can either be specified as total number of atoms, fraction of atoms, or percentage of atoms in the specified group. Note that if the concentration is expressed as either a fraction or percentage of atoms, then the total number of atoms to be simulated must also be provided. This is done by specifying the quantity using the natoms parameter. For a basic run, the mass and chemical potential of each atom can simply be set to zero as these parameters are mainly for use by specific potentials and fitness calculations respectively.

The suggested parameters can be entered into a text file using the format shown below:

structure = Cluster/Crystal/Defect

optimizer\_type = Random /BH/SA/GA

atomlist = [(Type of atom 1, concentration of atom 1, mass of atom 1, chemical potential of atom 1),

(Type of atom 2, concentration of atom 2, mass of atom 2, chemical potential of atom 2),

(Type of atom 3, concentration of atom 3, mass of atom 3, chemical potential of atom 3),

…]

natoms = <integer number>

SolidFile = <filename for solid structure>

The code will parse the input file into a python dictionary once it is read. Alternatively, a user could also initiate an optimization using a python dictionary without the need for an input file. This input approach will be explored further in the [Input and Output Syntax](#_Functionalities) section of the document. If one wanted to simulate a 20 atom copper and gold nanocluster by basin hopping using the input file format shown above, the input file would look like this:

structure = Cluster

optimizer\_type = BH

atomlist = [(‘Au’, 10, 0, 0), (‘Cu’, 10, 0, 0)]

This would optimize a structure including 10 gold atoms and 10 copper atoms.

To run this simulation, one would simply open a terminal window and type the following:

python <path\_to\_StructOpt\_Optimizer\_Script> <path\_to\_input\_file\_and\_name>

Alternatively, one could also type the following to execute the program interactively:

python

from structopt import Optimizer

A = Optimizer(<input\_file\_path\_and\_name>)

A.run()

The genetic algorithm will then execute using the default ASE built-in Leonnard-Jones potential. The default configuration will initialize a serial algorithm. Note that running a file using only these inputs will activate the default settings and will not provide fully optimized results. Additional parameters such as number of steps, metropolis temperature, and a correct potential should be provided in order to get more converged results. However, the simple input shown here is a good way to test that the program is operational and the environmental variables are set correctly. For more information concerning how to include additional parameters please refer to the [Functionalities](#_Functionalities_1) section.

The algorithm will output the following file and folder structure as a default:

Output-rank0.txt

Output-rank0-XXXX\_XXXX\_XXXXXX.log

Output-rank0/

Bests-fitnesses-Output.txt

Bests-Output.xyz

Indiv00.xyz

StructureSummary.txt

Summary-Output.txt

Optimizer-restart-file.txt

Restart-files/

Reload-bestsXX.txt

Reload-indivXX.txt

The “Output-rank0.txt” file contains the details of the algorithm performed including input parameters, applied moves and crossovers, summary of the energy calculations, and performance overview. It is the primary output for the program and will be updated after the energy calculations. The Output-rank0-….log file provides a more detailed analysis of the progression of the program including parameter settings, structure calculations, error reporting, and more. The Output-rank0 folder contains many files of interest to the user. The Best-fitnesses-<filename>.txt and Bests-<filename>.xyz files contain the record of the top structures and a list of their fitnesses from the run. These will be updated every five iterations of the algorithm. The primary structure output (indiv00.xyz, indiv01.xyz,…) contains the individuals in the population from each iteration (or generation) of the algorithm. For instance, if there are five structures in a genetic algorithm run, there will be five primary structure output files labeled 00-04. Each file will contain multiple structures namely one structure for each generation. The structures are ordered based on their relative fitnesses with the best structure in each generation being written to the 00 file, the second best structure to the 01 file, the third best structure to the 02 file and so on. Viewing the contents of the indiv00.xyz file will allow the user to view the evolution of the best structure in the generation. A summary of the structures file (StructureSummary.txt) contains additional information about the structures in each generation. A summary of the fitness for the structures in the population throughout the evolution of the run is contained in the summary output file (Summary-Output.txt). The files in the Output folder will be updated at the end of each generation. The restart files are created for the purpose of continuing an optimization from the point where it was last completed. For example, to restart the example program after it has completed running, the user should open the Optimizer-restart-file.txt and change the following lines:

'maxgen':10,

'convergence': True,

Then the program can be restarted using the following command:

python <path\_to\_StructOpt\_Optimizer\_Script> <path\_to\_Optimizer-restart-file.txt>

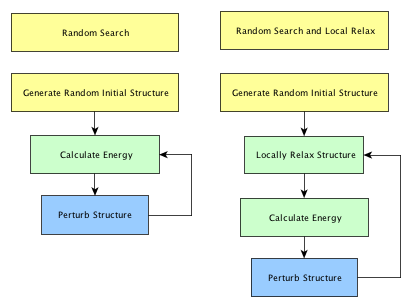
A more detailed description of each of these files and additional output options can be found in the Output Files section of this document.

# Algorithm Overview

StructOpt is currently capable of four broad optimization options: random search, basin hopping, simulated annealing, and genetic algorithm. The choice of applied optimizer can be set using the optimizer\_type input parameter, which will select a few key default parameters. Alternatively, setting these various parameters independently can allow users to mix and match various methods. The following subsections highlight the basics of each standard approach.

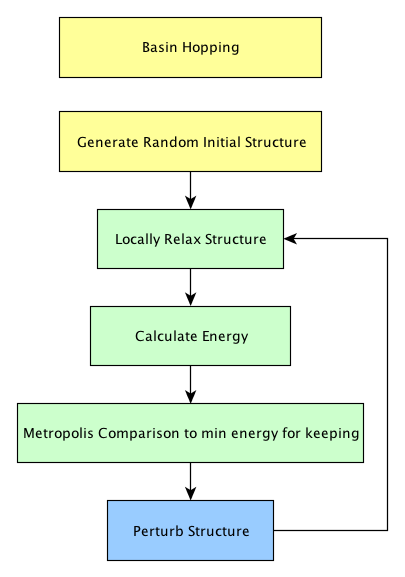
## Random Search

A random search optimization involves random or sometimes system-specific alterations to the structure to attempt to optimize globally. The random search will alter individual atoms or groups of atoms repeatedly until a new lower local energy is identified. Typically, random search methods will involve some sort of local relaxations in order to improve their efficiency. For information on how to use random searching and examples of strong applications of this technique, users should reference the research of Dr. Chris J. Pickard.



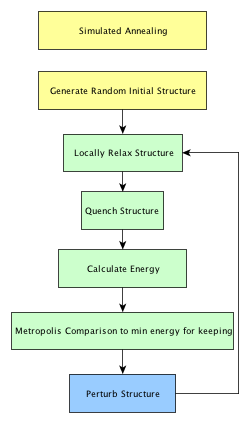
## Basin Hopping

A basin hopping minimization is similar to the random search minimizer with the exception that it includes a metropolis criterion that allows even those structures with higher energy to persist and be re-perturbed. It allows for a more systematic search of the potential energy landscape. For more information and examples of strong applications of basin hopping, users should refer to the works of Dr. David Wales.



## Simulated Annealing

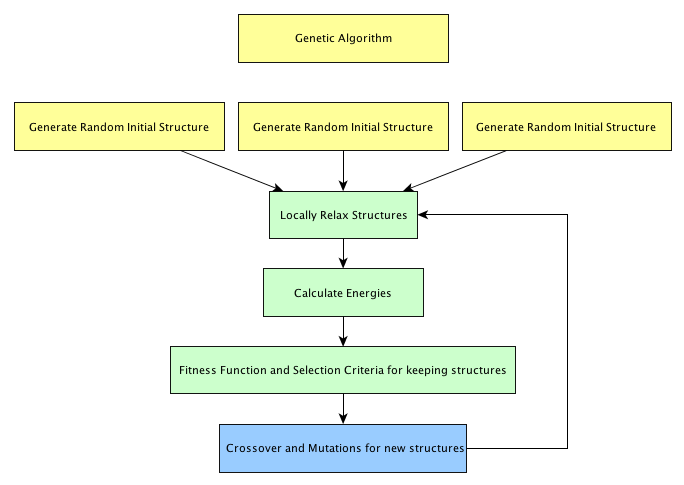
A simulated annealing optimization type employs a quench minimization between random moves. The quench step essentially heats the structure and then slowly cools it allowing the atoms to overcome some of the potential energy barriers to their motion. It also uses a metropolis selection criterion that allows even those structures that are less stable to survive to be re-quenched. For more information and examples of strong applications of the simulated annealing technique, users should refer to the works of Dr. J. Christian Schön and Dr. M. Jansen.



## Genetic Algorithm

Unlike the previously mentioned algorithms, a genetic algorithm utilizes a population of structures rather than a single individual. This allows for structures with a range of stabilities to exist and be perturbed in order to find the global minima. The genetic algorithm also makes use of crossover schemes that enable information such as atom positions and types to be exchanged between individuals. The idea behind this being that structures that are most stable are stable because of their particular arrangements of atoms and that if sections of those arrangements are exchanged with less stable structures a more stable structure may result. In addition to crossovers, the genetic algorithm also uses random moves for a select percentage of the structures in the population to help ensure the diversity of the structures in the population. For additional information on genetic algorithm optimization of structures, please refer to the works of the following authors:

* Crystals
  + Artem Oganov
  + Alex Zunger
  + Scott Woodley
  + Richard Catlow
* Clusters
  + Roy L. Johnston
  + Bernd Hartke
  + David Deaven
* Surfaces
  + Cristian V. Ciobanu
  + Kai-Ming Ho



## Parallelization Schemes

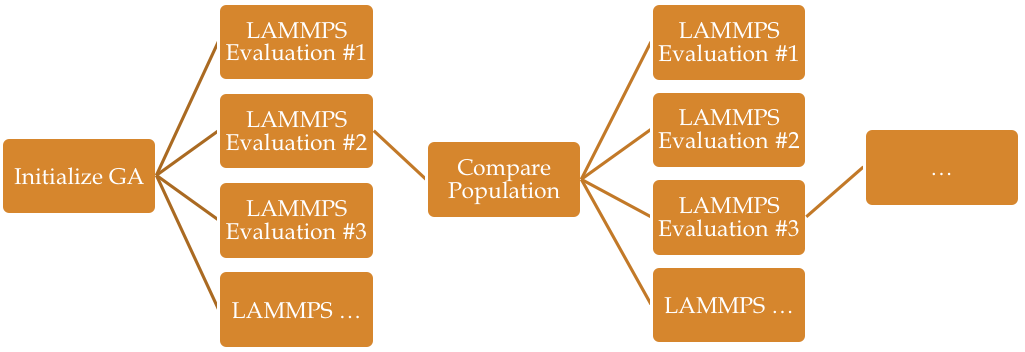
In addition to the standard serial operations described in the subsections above, StructOpt contains a few options for parallelizing structure optimizations. These methods are primarily geared towards a genetic algorithm optimization as the other optimization schemes are inherently serial. Note, to use these techniques you must have MPI4PY (<http://mpi4py.scipy.org/docs/usrman/index.html>) installed. For the parallel schemes using MPI4PY, it is recommended that only one node be used for the LAMMPS calculation as multi-node calculations may confuse the parallelization scheme and result in several structures being improperly calculated and removed from the population.

### Serial:

In the standard serial scheme, energy calculations are performed one after another and the resulting fitnesses are compared at the end of all of these calculations. In this scheme, the built-in python threading and subprocess modules are used to initialize LAMMPS and track when it has completed.

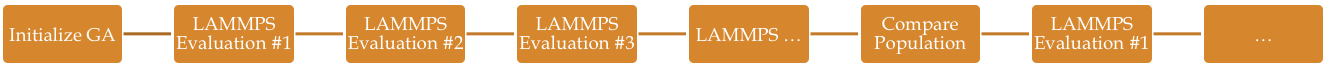
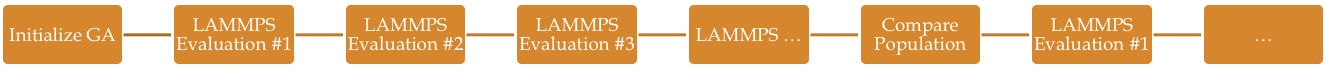
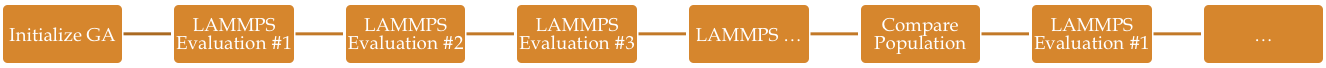
### Standard Parallel:

In the standard parallel scheme, energy calculations are divided among slave processors with the comparisons, mutations, and crossovers performed on the master processor.



### Island Method:

In the island method algorithm, each processor receives a separate optimization case and every few generations the processors exchange low energy structures.



### MAST:

The MAST algorithm allows the optimizations to be performed on a queue submission basis. Each energy evaluation is submitted to the queue and the MAST algorithm tracks their completion before continuing with the comparisons and operations. This enables optimizations using VASP or LAMMPS on multiple processors, which may take a significant amount of time to complete.

# Input and Output Syntax

## Input Options

StructOpt was designed to be a standalone program as well as an integrated program with other python codes. As such there are a few different ways in which the input parameters can be constructed. Either an input file or a python dictionary is an acceptable input format.

Regardless of the method of input, the input parameters should be structured as follows:

structure = Cluster/Crystal/Defect

atomlist = [(Type of atom 1, concentration of atom 1, mass of atom 1, chemical potential of atom 1),

(Type of atom 2, concentration of atom 2, mass of atom 2, chemical potential of atom 2),

(Type of atom 3, concentration of atom 3, mass of atom 3, chemical potential of atom 3),

…]

natoms = <integer number>

SolidFile = <filename for solid structure>

parameter1 = <options>

parameter2 = <options>

parameter3 = <options>

…………….

The parameters shown here are the minimum parameters required to run the program with the exception of the SolidFile parameter which is only required for the Defect case. For additional parameters and their functions, see the [Functionalities](#_Functionalities_1) section.

There are three primary methods for executing the program: command line with input file, interactive python session with input file, and python script without input file.

### Submitting via command line with input file

In order to use this submission format, the input parameters must first be written to an input file in the working directory. To run the simulation, one would simply open a terminal window in the working directory and type the following:

python <path\_to\_StructOpt\_Optimizer\_file> < input\_file\_name>

This will execute the program using python with the parameters specified in the input file provided.

### Submitting via interactive python session with input file

In order to use this submission format, the input parameters must first be written to an input file in the working directory. To run the simulation, one would simply open a terminal window in the working directory and type the following:

python

from structopt import Optimizer

A = Optimizer(<input\_file\_name>)

A.run()

This input will open an interactive python session, load the StructOpt python module, initialize a StructOpt program with the parameters provided in the input file, and execute the StructOpt program.

### Submitting without an input file

Alternatively, the input parameters can be provided as a python dictionary either in an interactive python session or in a python script. To run the simulation from an interactive python session without generating an input file, one would simply open a terminal window in the working directory and type the following:

python

from structopt import Optimizer

parameters = {‘structure’ : ‘Cluster/Crystal/Defect’,

‘atomlist’: [(Type of atom 1, concentration of atom 1, mass of atom 1, chemical potential of atom 1),

(Type of atom 2, concentration of atom 2, mass of atom 2, chemical potential of atom 2),

(Type of atom 3, concentration of atom 3, mass of atom 3, chemical potential of atom 3),

…],

‘natoms’ : <integer number>,

‘SolidFile’: ‘<filename for solid structure>’

‘parameter1’ : <options>,

‘parameter2’ : <options>,

‘parameter3’ : <options>,

……………. }

A = Optimizer(parameters)

A.run()

This format allows the user the greatest flexibility as this format can also be used to generate python scripts that execute multiple jobs by looping over a variety of input parameters. The python script would look identical to the above format with the exclusion of the first python command. The script could then be executed from the working directory as a typical python script:

python <python\_script\_name>

### Submitting Through MAST

StructOpt is also available as a part of the MAterial Simulation Toolkit (MAST), which allows users to submit multiple optimizations or utilize the queuing system to submit longer optimizations (such as VASP calculations or multiprocessor LAMMPS calculations).

To perform a StructOpt optimization with MAST, one can either set the StructOpt input file path in the MAST input file or the user can add the parameters explicitly in the MAST input file. Note that any StructOpt parameters listed in the MAST input file override any parameters in the given input file.

Below is an example for submitting a single StructOpt optimization through MAST using a StructOpt input file.

**StructOpt Input file:**

structure = Cluster

atomlist = [(‘C’,10,0,0)]

**MAST Input file:**

$mast

system\_name StructoptTest

$end

$structure

coord\_type fractional

begin lattice

3.5 0 0

0 3.5 0

0 0 3.5

end

begin coordinates

Si 0.0000000000 0.0000000000 0.0000000000

end

$end

$ingredients

begin ingredients\_global

mast\_nodes 1

mast\_multiplyencut 1.5

mast\_ppn 1

mast\_queue default

mast\_exec python //home/bin/structopt/structopt/Optimizer.py

structopt\_input\_file //home/StuctOpt\_Input\_file.txt

calc\_method LAMMPS

mast\_program structopt

mast\_write\_method write\_singlerun

mast\_ready\_method ready\_singlerun

mast\_run\_method run\_singlerun

mast\_complete\_method complete\_singlerun

mast\_update\_children\_method give\_structure

end

$end

$recipe

recipe\_file StructOptTest.txt

$end

**MAST Recipe file:**

Recipe StructOptTest  
cluster\_opt1

Alternatively, specifying the parameters only in the MAST input file as in the example below performs the same optimization.

**MAST Input File:**

$mast

system\_name StructoptTest

$end

$structure

coord\_type fractional

begin lattice

3.5 0 0

0 3.5 0

0 0 3.5

end

begin coordinates

Si 0.0000000000 0.0000000000 0.0000000000

end

$end

$ingredients

begin ingredients\_global

mast\_nodes 1

mast\_multiplyencut 1.5

mast\_ppn 1

mast\_queue default

mast\_exec python //home/bin/structopt/structopt/Optimizer.py

structure Cluster

atomlist [('C',10,0,0)]

calc\_method LAMMPS

mast\_program structopt

mast\_write\_method write\_singlerun

mast\_ready\_method ready\_singlerun

mast\_run\_method run\_singlerun

mast\_complete\_method complete\_singlerun

mast\_update\_children\_method give\_structure

end

$end

$recipe

recipe\_file StructOptTest.txt

$end

**MAST Recipe File:**

Recipe StructOptTest  
cluster\_opt1

The first section of this input file contains information on a structure for MAST. Because StructOpt does not use an input structure, this section is not important and can be set to anything as long as the MAST input parser can still parse it. The second section is the ingredients section. This is where all of the input parameters for StructOpt will be listed. In this section, the mast\_exec line should point to the python path and the Optimizer.py file. Essentially, MAST uses the submission from command line input method to communicate with StructOpt in this example. MAST will generate a python dictionary input file for StructOpt from the parameters given in the MAST input file. Unlike in the standard input file, the parameters listed in the MAST input file do not need an equal sign. This input file will simply use MAST to run StructOpt.

Alternatively, MAST can be used to control the evaluations of each generation by separately submitting each structure to the queue. This will allow a basic parallelization of the optimization without the need for mpi4py. Below is a sample input script to run this method.

**MAST Input File:**

$mast

system\_name StructoptTest

$end

$structure

coord\_type fractional

begin lattice

3.5 0 0

0 3.5 0

0 0 3.5

end

begin coordinates

Si 0.0000000000 0.0000000000 0.0000000000

end

$end

$ingredients

begin ingredients\_global

mast\_nodes 1

mast\_multiplyencut 1.5

mast\_ppn 1

mast\_queue default

mast\_exec python //home/bin/structopt/structopt/Optimizer.py

PAIR\_STYLE eam

MINIMIZE 1e-5 1e-5 5000 10000

MIN\_STYLE cg

MIN\_MODIFY line quadratic

potential\_file /home/potentials/Au\_u3.eam

exec\_mast //share/apps/lammps\_280312

structure Cluster

optimizer\_type GA

atomlist [('Au',1,196.9665,0.0)]

natoms 10

nindiv 5

maxgen 5

calc\_method MAST\_LAMMPS

mast\_program structopt

mast\_write\_method write\_singlerun

mast\_ready\_method ready\_singlerun

mast\_run\_method run\_singlerun

mast\_complete\_method complete\_singlerun

mast\_update\_children\_method give\_structure

end

$end

$recipe

recipe\_file simple\_structopt.txt

$end

**MAST Recipe File:**

Recipe StructOptTest  
cluster\_opt1

In this input file, the structure section is ignored and the ingredients section contains the parameters for running both StructOpt and LAMMPS. In order to use MAST to calculate the structures, the calc\_method parameter must include MAST in the value. Additionally, the user also needs to specify the path to the LAMMPS executable file using the exec\_mast parameter. A similar structure is used to run VASP with MAST. The exceptions would be using the VASP parameters in the ingredients section, the path to the VASP executable, and including VASP in the calc\_method parameter.

Users can also use MAST to run multiple StructOpt optimizations by taking advantage of built in looping to run optimizations with different parameters settings such as number of atoms, random number seeds, convergence criterion, etc. Please refer to the MAST documentation for information on how to set up appropriate input files and recipes to take advantage of these tools.

MAST can also be used to execute both self-contained StructOpt calculations and queue submissions. For instance, if a user wanted to run a few generations with LAMMPS and a few generations with VASP, they could use MAST to set up and execute the workflow. An example of such an input file is shown below.

**MAST Input File**

$mast

system\_name StructoptTest

$end

$structure

coord\_type fractional

begin lattice

3.5 0 0

0 3.5 0

0 0 3.5

end

begin coordinates

Si 0.0000000000 0.0000000000 0.0000000000

end

$end

$ingredients

begin ingredients\_global

mast\_nodes 1

mast\_multiplyencut 1.5

mast\_ppn 1

mast\_queue default

mast\_exec python //home/amyk/bin/structopt/structopt/Optimizer.py

potential\_file /home/amyk/potentials/Au\_u3.eam

structure Cluster

optimizer\_type GA

atomlist [('Au',1,196.9665,0.0)]

natoms 10

nindiv 5

maxgen 5

calc\_method LAMMPS

mast\_program structopt

mast\_write\_method write\_singlerun

mast\_ready\_method ready\_singlerun

mast\_run\_method run\_singlerun

mast\_complete\_method complete\_singlerun

mast\_update\_children\_method give\_structure

end

begin vasprun

ISTART 0

ICHARG 2

PREC Accurate

LREAL Auto

ISIF 3

ENCUT 425

EDIFF 1E-04

ALGO Normal

NSW 99

EDIFFG 1E-03

IBRION 2

ISMEAR 1

SIGMA 0.2

NPAR 1

LWAVE .FALSE.

LCHARG .FALSE.

exec\_mast //share/apps/vasp

restart True

calc\_method MAST\_VASP

maxgen 1

end

$end

$recipe

recipe\_file StructOptVasp.txt

$end

**MAST Recipe File:**

Recipe StructOptVasp  
cluster\_opt1

cluster\_opt2 (vasprun)

Again, in this file the first section with the structure information is ignored. The second section includes parameters for each of the two separate runs. The first run section includes the parameters for the run using LAMMPS through StructOpt and section run section includes the parameters for both VASP and StructOpt. Note, the parameters shown here are just for example and have not actually been tested.

### Restarting a Run

There are two main ways to restart a previous calculation. The simplest method is to simply set the restart flag equal to True in the input file/dictionary. If this is the case, the program will look for a previous optimization run in the current working directory and attempt to load the last structures given in the individual files in the run folder with the filename specified in the input parameters or the default filename. For a defect in a bulk structure, the user can specify a number of restart integers. This number should correspond approximately to the number of atoms in the set 1 and set 2 for the individuals in a population. It should be noted that because this restart method utilizes the ‘xyz’ file format outputs, it could not be used to restart a crystal structure optimization, as the ‘xyz’ format does not include any information on the cell parameters. This method is the simplest way for users to seed a population or manually add structures to a population, as it will essentially start a new optimization from the structures given in the output folder. However, it should be noted that this method generates an entirely new optimizer class object from the input parameters with no memory of any previous optimization. Therefore, information such as the best structures list will be lost as well as previous runtimes and genealogy information.

The second method uses the restart optimizer file and the structures listed in the restart files folder in the output folder. This method is perhaps the best method to use if the user simply wants to extend a previous optimization a few more generations. Essentially, the restart optimizer file in the output folder contains all of the information contained in the optimizer object for the previous run including the names of the output files. To run a few more generations, the user would simply need to increase either the max\_gen parameter or the req\_rep parameter listed in this file depending on the specified convergence scheme. If the user wishes to change any other features of the optimizer, they must do so in this restart optimizer file. Note that the file is in the format of a python dictionary and will be loaded as such by the program when run. It should also be noted that the optimizer restart file contains paths to various output files and structures for the population. This means that if the user moves the output files from the previous optimization, they will need to change the paths to these files in the optimizer restart file.

## Output Files

The output files consist of the standard output, the structural output, the summary file, and the optional output files. The general output structure consists of the standard output file and a folder containing all of the other output files. All output files are updated at the end of each generation as it is completed.

### Standard Output

The standard output contains all of the details of the StructOpt execution. The name of the output file is set by the ‘Filename’ parameter. By default, it is a text file named ‘Output’ that will be generated in the directory from which the program is executed. It will include a description of all of the input parameters for the specific StructOpt run. This is particularly useful for runs that utilize a python input rather than an actual input file. The standard output includes information about the crossovers, mutations, fitness and energy evaluations, and predators that are applied every generation. Additionally, the information such as energy, fitness, and basic configuration data for the individuals that survive for each generation is output in the output file. Basically, all of the important information for the optimization is included in this file.

### Structural Output

One of the primary strengths of the genetic algorithm approach is the ability to generate not only the fittest structure but also other high fitness structures. As such, the StructOpt code automatically outputs the entire population for every generation. It will generate one xyz file for each structure in the population. The number of structures in the population is set by the Number of Individuals (nindiv) parameter. The files will be titled ‘indivXX.xyz’ where XX will be a number from 00 to the number of individuals in the population. Each file will keep track of a particular structure rank in a population. For instance, if the optimization includes a population of five individuals, then the following files will be generated: ‘indiv00.xyz’, ‘indiv01.xyz’, ‘indiv02.xyz’, ‘indiv03.xyz’, and ‘indiv04.xyz’. If the optimization is run for 10 generations, then each file will contain 10 separate structures. The structures will be ordered by their relative rank in the population for a given generation. This means that file ‘indiv00.xyz’ will include the structure with the best fitness for the given generation, file ‘indiv01.xyz’ will include the structure with the second best fitness, file ‘indiv02.xyz’ will includes the structure with the third best fitness for the structures in the generation. This way viewing the structures in a file can be seen as viewing the general evolution of the structure optimization. It should be noted that the structures written to these files represent the individuals that have survived the predator and natural selection processes. This means they do not necessarily represent all of the solution space searched by the algorithm as some structures that are generated from crossovers and mutations that are higher fitnesses are removed before being written to the output files.

### Summary File

The summary file outputs the most pertinent statistics for the StructOpt execution. This file will list the maximum fitness, minimum fitness, average fitness, standard deviation of the fitnesses, and completion time for each generation. It has the following format:

<Output Filename>

Defect Run Pure Bulk Energy per Atom: <Energy> *(optional for Defect case)*

Natoms <Total number of atoms>

Generation Fitmin Fitavg Fitmedium fitmax std time

XXX XXX XXX XXX XXX XXX XXX

XXX XXX XXX XXX XXX XXX XXX

XXX XXX XXX XXX XXX XXX XXX

…

It will be written to a file named:

Summary-<Output\_file\_name>.txt

There are a few different formatting options available for the summary file that can be customized using the output\_format input parameter. This allows the user to specify whether the output statistics should be given in the form of population fitnesses, energies, energy per atom, surface energies, or formation energies.

### Log File

The log file is generated upon initiating the Optimizer class object. Similar to the standard output file it includes lots of information that is pertinent to the optimization run performance. It will keep track of parameter settings, any errors during the execution, and can be used to track of where the algorithm is at any given point during the execution. For multiprocessor runs, a separate log file will be created for each processor.

### Restart Files

The restart files include the optimizer restart file and a folder containing the individual files for the structures in the population and best structures in the latest generation. These files are used if the user would like to run the optimizer for more generations or make alterations. See the Restarting a Run section of this document for more information on how these files are used and how they can be manipulated to serve the needs of the user.

### Optional Output Files

#### Genealogy File

The optional genealogy output file provides information on the history of the structures in the population. This allows the user to identify how the final structures were generated, which mutations proved the most useful, how successful the crossovers were, etc. While the actual genealogy output file is rather simplistic, the read\_output program will generate a visual representation of this file. In general, each line in the genealogy file corresponds to a generation. The population is then listed with the best individual at the left and the worst individual at the right. The individuals are shown in relation to the individuals of the previous generation. Crossovers between individuals are represented by plus signs. Mutations are represented by an ‘m’ and then typically one or two letters and possibly a number representing the type of mutation performed and how many atoms it affected. See the mutation part of the [Functionalities](#_Functionalities_2) section for more information on these mutation codes. The first generation is always simply shown as an ascending list.

For example, consider the following three generations:

0 1 2 3 4

(2+4) (0+2) 1 3 1mLA1

0 (3+4)mGR2 1 (2+1)

The first row indicates the first generation. It is shown as an ascending list from 0 to 4 indicating that there are five structures in the population. The second row represents the second generation. The lowest fitness zero structure for the second generation is shown as the result of a crossover between the previous generation’s second lowest fitness structure and fourth lowest fitness structure. Similarly, the lowest fitness first structure is a combination of the previous generation’s zero lowest fitness structure and second lowest fitness structure. The lowest fitness second and third structures are simply the previous generation’s first lowest fitness structure and third lowest fitness structure. This indicates that these individuals survived the predator. Finally, the fourth lowest fitness structure is a mutated version of the previous generation’s first lowest fitness structure. In this case, this structure represents a lattice alteration mutation of a single atom from the structure in the previous generation. It should also be noted, that it is typical in a real simulation for the second generation to simply be a reordered first generation as often the first generation is listed as the order in which the structures were generated not necessarily the order sorted by fitness. The third row now represents the third generation with respect to the second generation structures. In this case, the zero lowest fitness structure for this generation is the zero lowest fitness structure of the second generation, which was a crossover of the second and fourth lowest fitness structures of the first generation. The first lowest fitness structure of the third generation shows an example of a compound structure. In this case, the structure is the result of a crossover between the third and fourth fitness structures and then a group rotation mutation of two atoms in that combined structure.

The genealogy file can be extremely useful for evaluating an optimization run as it can be used to indicate the selective pressure in the optimization. For instance, if mutations and crossovers are only very rarely effective, it can indicate that the optimization parameters may be set too tightly to allow the algorithm to search the entire solution space or vice versa the parameters may be set too loosely to allow the best structures to survive for long enough to pass on their better properties.

#### All Energy File

This optional file output writes the raw energies of every structure in a generation. This is the energy as it is received from the energy evaluation code using either a molecular dynamics simulation such as LAMMPS or a DFT simulation such as VASP. Outputting this file can be useful for users who are interested in evaluating the raw energies either to ensure the proper functioning of these separate codes or for later post-processing of the data.

#### Individual Defect Structures

This optional file output is for use with Defect structure optimization. These outputs are mainly used for debugging and for optimizations in large super cells. It allows the user to monitor the atoms being written to region 1 and region 2 throughout the evolution of the structures, which can be useful in determining why crossovers and mutations may not be particularly effective. For more information on the definition of these regions, see the description of a Defect structure optimization in the Functionalities section of this document.

## Post-Processing

Current post-processing options are currently limited and are often left to the discretion of the user. There are three main options currently available. These include the standard performance analysis, the genealogy analysis, and the lattice concentration analysis. The standard performance analysis post-processing plots data for fitness distribution as a function of generation, a plot of all the energies of the population throughout the optimization, a plot of the minimum fitness fingerprints, a plot of the distances between fingerprints for a population, a plot of the timing for each generation, and a plot of the successful mutations over the optimization. The genealogy tree produces a graphical display of the genealogy of each structure in the optimization through the evolution. Note, for longer runs, the genealogy tree can be quite large. Users should use caution when using this output option as it can produce a very large ‘png’ file and require a large amount of memory. The third option is for ‘Defect’ optimizations. It provides a method for evaluating lattice atom concentration versus interstitial atom concentration by comparing the output structures to the original crystal structure. This data will be output to a file called “LatticeConcentration.txt” in the output directory.

# Functionalities

Listed in this section are the details of the main functions and parameters utilized by StructOpt.

## Primary Parameters

### Structure

The structure input parameter specifies what type of structure will be optimized. The type parameter can be set to one of the following: ‘Cluster’, ‘Crystal’, or ‘Defect’. This input parameter is **required** to execute a given simulation.

#### Cluster

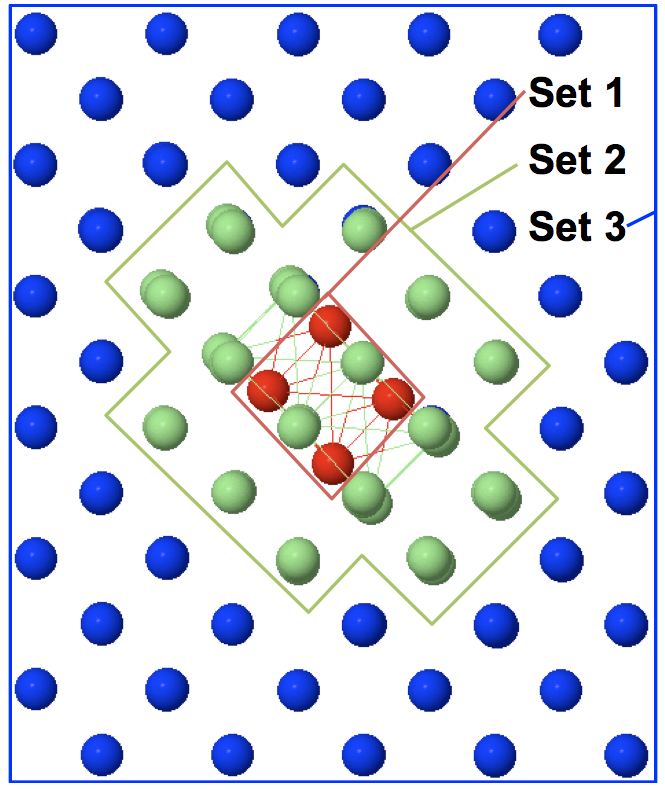
If the type of simulation is set to ‘Cluster’, then the StructOpt simulation will assume a non-periodic cluster of atoms is desired. This is ideal for analyzing stable nano-cluster configurations. This form of algorithm will optimize all of the positions and types of atoms in the simulation. It should be noted that most energy evaluation programs such as LAMMPS and VASP require periodic boundary conditions. To account for this the cluster is embedded in a large simulation box. The size of this box can be set using the Size of Large Box (large \_box\_size) parameter.

#### Crystal

If the simulation type is set to ‘Crystal’, then the StructOpt simulation will assume a periodic arrangement of atoms. In addition to optimizing the position and types of atoms in the structure, this structure optimization will also attempt to optimize the lattice parameters for the structure.

#### Defect

If the type of simulation is set to ‘Defect’, then the StructOpt simulation will assume a periodic arrangement of atoms with a localized embedded defect. This type of optimization will divide the problem out into three main atom sets depicted in the figure below. The crossovers and mutations will only be applied to set 1 (red) and set 2 (green).



### Optimizer Type (optimizer\_type)

This is an optional input parameter that can be used to select an array of default values. Choices include Random, BH, SA, or GA and are reviewed in the Algorithm Overview section of this document. The default parameters for each setting are described here. It should be noted that any additional user provided inputs always override these parameters.

#### Random

This type of optimizer employs a random search. It will apply the following defaults:

nindiv = 1

genealogy = False

number\_of\_bests = 100

cxpb = 0.0

mutpb = 1.0

natural\_selection\_scheme = best

#### Simulated Annealing (SA)

This type of optimizer employs a simulated annealing approach. It will apply the following defaults:

nindiv = 1

genealogy = False

number\_of\_bests = 100

cxpb = 0.0

mutpb = 1.0

natural\_selection\_scheme = metropolis

#### Basin Hopping (BH)

This type of optimizer employs a basin hopping approach. It will apply the following defaults:

nindiv = 1

genealogy = False

number\_of\_bests = 100

cxpb = 0.0

mutpb = 1.0

natural\_selection\_scheme = metropolis

#### Genetic Algorithm (GA)

This type of optimizer employs a genetic algorithm. It will apply the following defaults:

nindiv = 10

genealogy = True

number\_of\_bests = max(100, parameters[‘nindiv’])

algorithm\_type = 'lambda+mu'

cxpb = 0.8

mutpb = 0.15

natural\_selection\_scheme = tournament

### Atom List (atomlist)

The atomlist input parameter specifies the number, type, mass, and chemical potential of the atoms to be used in the simulation. This input parameter is **required** to execute a simulation. The atomlist must be formatted as follows:

atomlist = [(Type of atom 1, concentration of atom 1, mass of atom 1, chemical potential of atom 1),

(Type of atom 2, concentration of atom 2, mass of atom 2, chemical potential of atom 2),

(Type of atom 3, concentration of atom 3, mass of atom 3, chemical potential of atom 3), …]

Formatting the atomlist improperly (including leaving out one of the four parameters for an atom) will result in an error. In general, it is good practice to set the parameters that will not be used, such as mass or chemical potential, to zero.

### Number of Atoms (natoms)

The natoms parameter specifies the number of atoms to be used in an individual. It will be automatically set by the [atom\_list](#_Atom_List) parameter if it is not otherwise specified. It is a **required** input parameter if the atom\_list concentrations are given as fractions or percentages. In general, it is also a good practice to specify this parameter any time a simulation involving vacancies is used. This parameter should be provided as a python integer value. It should be noted that for some simulations the number of atoms in an individual might not necessarily be fixed depending on the other parameters specified for the simulation. However, the natoms parameter does help to set the initial population.

### Number of Individuals (nindiv)

The nindiv parameter specifies the number of clusters or individuals to be used in a simulation. For all optimization styles other than a genetic algorithm, the nindiv parameter is set to 1. For a genetic algorithm, it has a default value of five. However, typical values of the nindiv parameter for a genetic algorithm can vary between 10 and 50 depending on the type of system being optimized. The nindiv parameter should be provided as an integer value. In an ideal world, the value of this parameter would be very large (close to infinite), as this would allow the simulation to obtain the highest level of diversity. However, this would create a prohibitively large computational time. Instead, it is common to use lower quantities of individuals along with some means of diversity control.

## Output Parameters

### Filename

The filename parameter specifies the name for the output folder and files for the simulation. The default value for this parameter is ‘Output’. The parameter will be included in all of the output files and folders in order to keep information from the optimization identifiable.

### Genealogy

The genealogy parameter specifies whether or not the simulation should output the genealogy file. This parameter may be either true or false. The default value is false, which will not output a genealogy file for the simulation.

### Output Format (output\_format)

The output format parameter specifies the format to be written to the summary file. The default value for a ‘Defect’ type simulation is ‘formationenergy’. The default value for all other types of simulations is ‘totalenergy’.

### All Energy File Output (allenergyfile)

This output option specifies whether or not to write a file containing the raw energies of every structure in a population for each generation. This parameter may be either true or false. The default value is false, which will not output an all energy file for the simulation.

### Best Individuals Output (best\_inds\_list)

This output option allows the algorithm to generate a list of the highest fitness individuals during the evolution. This parameter may be either true or false. The default value is false, which will not output a best individual file for the simulation.

### Number of Bests (Number\_of\_Bests)

This optional parameter specifies the number of individuals to store in the list of best individuals over the length of the optimization. The default value is typically 100 structures.

### Write Individual Defect (indiv\_defect\_write)

This optional output parameter is for use with a Defect structure optimization and specifies whether or not to output additional files for each individual in the population that depict only regions 1 and 2 for a given structure. See the Output Files section of this document for more information on this parameter.

### Output Vacancies (vacancy\_output)

This option is primarily for use with a Defect structure optimization and specifies whether or not to include an atom in a vacant lattice position. This can be useful for identifying dumbbell structures or identifying interstitial clusters in relation to lattice positions. For vacancy defect simulations, this output can be very useful.

## Post-Processing Parameters

### Lattice Concentration Output (lattice\_concentration)

The lattice concentration parameter evaluates the concentration of atoms on lattice sites after a ‘Defect’ type simulation. It is useful if the concentration of atoms of a specific type in the defect versus on the lattice is in question.

### Standard Post-Processing (postprocessing)

The standard post-processing flag will allow for the program to conduct some basic post-processing analysis on the results of the optimization. The exact post-processing that is output will depend on the flags used for input. This includes a plot of all of the energies and energy per atom of the structures that survive the natural selection versus the generation, a plot of maximum, minimum, and medium fitness versus generation, a plot of the computing time for each generation, a plot of the best fitness structure fingerprint function for each generation, a plot of the fingerprint distances for each structure in the population versus the generation, and a plot of the successful mutations.

### Genealogy Tree (genealogytree)

This parameter will determine whether or not the program should generate a genealogy tree. The genealogy tree is the visual representation of the genealogy file output. This function is still very much in development and should be used in caution. It will scale the size of its output plot based on the number of generations and number of individuals. Therefore for a large population or a large number of generations will result in a very large file size.

## General Algorithm Parameters

### Restart (restart)

The restart parameter allows the user to start a StructOpt simulation using previously run data. This parameter may be set to either true or false. The default value is false. Setting this parameter equal to true will allow the simulation to attempt to load previous structures as the initial population for a new run. Note that the input parameters that govern the generation of structures should be the same as in the original run. The simulation will look for a population of structures within a folder with the name provided in the filename parameter. If the simulation is unable to find or read the needed files, it will generate an error. If the simulation finds fewer files than specified in the [nindiv](#_Number_of_Clusters) parameter, it will reduce the number of individuals allowed in a population. Please refer to the Restarting a Run section of this document for more information on this parameter.

It should be noted that this parameter is fairly flexible and can be very useful for manually altering an optimization run. For instance, this parameter can be used if the user would like to seed a population with a particular type of structure or wants to test the relative stability of a structure with respect to other structures in an optimization. For more information on how to seed a population with a particular set of structures, please refer to the Seeding a Population section of this document.

### Number of Restart Atoms (restart\_ints)

This optional parameter is primarily for use in a Defect optimization and is used to identify those atoms belonging to region 1 and 2 from the previous simulation. This is an alternative to simply employing the defect identifier to redefine these new regions for the new simulation. The parameter defaults to 0 but is otherwise specified as an integer. If the value is set to 0, then the find defect scheme will be applied

### Seed (seed)

The seed parameter allows the user to specify a random number seed for the simulation. The default value will be a random integer number between 0 and 10. In order to ensure convergence, it can often be good practice to run a simulation with different random number seeds. There are no real suggestions for what this parameter should be set to, however, if the user is intending on conducting multiple separate optimization runs, she/he should ensure that this parameter is different between each run. In the Island Method algorithm, applying separate random number seeds for each processor generates the different populations.

### Forcing Control (forcing)

The forcing control parameter allows the user to specify how the algorithm should handle variable atom type cases. The default value for this parameter is ‘Concentration’. This value ensures that concentration of atoms within an individual will remain constant throughout the simulation. Such control ensures that structures that do not have the desired concentration of atoms will not waste computational resources. Optional values of this parameter include: ‘energy\_bias’ and ‘chem\_pot’. The idea behind theses two alternative options is to apply a penalty to structures with incorrect number or types of atoms to make them less favorable in a fitness evaluation rather than force the structure during the crossover and mutation procedures. These options are currently degenerate functions and should not really be used. This is primarily because of the computational cost in re-fining the structure using multiple crossover attempts and the cost of doing a full local energy minimization of a structure. The options are presented here and in the code to provide an alternative approach to controlling structure parameters. However, generally it is recommended that if the user is attempting to add a bias to the structure evaluation, she/he should do so in an new fitness function and simply set forcing control to ‘Off’.

### Parallel (parallel)

The parallel parameter allows the user to specify whether or not to conduct the simulation using multiple processors or computer nodes. This parameter may be either true or false. The default value is false, which will run the simulation in series. Note that to use this parameter, the user must first set up a parallel version of the program see the [Parallelization](#_Parallelization) section for more information on how to do this.

### Debug (debug)

The debug parameter allows the user to select sections of the code for debugging. This can be useful for evaluating new user added options. The default value for this parameter is ‘None’. To specify sections for debugging, the user should provide a list of strings. Available options for debugging include:

MA – Used to debug the main optimizer algorithm. Generally this is only used in serial mode.

SEL – Used to debug selection schemes

MU – Used to debug mutation schemes

CX – Used to debug crossover schemes

FIT – Used to debug fitness schemes

EE – Used to debug the energy evaluation algorithm

FD – Used to debug find defects function

SetCalc – Used to debug the setup calculator function

### Algorithm Type (algorithm\_type)

The algorithm parameter allows the user to select which type of algorithm to use. The default value for this parameter is lambda+mu. In this scheme, the parent structures compete with the offspring for available slots in the next population. The success of an individual is determined by the natural selection function (Natural Selection Scheme (natural\_selection\_scheme)).

An optional value for this parameter is ‘Island\_Method’. This optional value is used to specify a particular type of parallel algorithm. An Island Method algorithm establishes a genetic algorithm run with different random number seeds on each processor provided. Periodically, the algorithm will switch a certain number of its fittest individuals with those on another processor. In order to use the Island Method algorithm, the user should also specify the intervals at which to migrate the individuals between processors. This is accomplished through the ‘migration\_intervals’ parameter. The default value for this parameter is 5 generations. Therefore, every five generations the processors will exchange their best individuals as if in a ring. In other words, the best individuals on processor 1 will be given to processor 2, those on processor 2 will be given to those on processor 3, and so on until the last processor gives its best individuals to processor 1. The ‘migration\_percent’ parameter specifies how many individuals to transfer between the processors. The default value for this parameter is 0.05 or 5% of the total population will migrate. Note that use of this method requires the MPI4PY module. See the Parallel Installation section of this document for additional information on parallel installations of StructOpt. For additional information on how the island method is implemented in StructOpt, refer to the Parallelization Schemes section.

The third value for this parameter is ‘lambda,mu’. This parameter sets the type of genetic algorithm to a (λ,μ) scheme rather than the default (λ+μ). In this scheme, the offspring will replace the parents with the parents only surviving if there are too few offspring to replace them. This algorithm has a default elitism function that always ensures that the lowest energy structure survives. In order to implement an island method (λ,μ) scheme the user should input the following: ‘Island\_Method-lambda,mu’. Essentially, the algorithm will search for both the flags ‘Island\_Method’ and ‘lambda,mu’.

### Migration Intervals (migration\_intervals)

This optional parameter is for use with the Island method algorithm type. It is used to specify the number of generations between a migration exchange event across the different processors involved in a simulation. Higher quantities of this parameter will reduce the amount of interaction between the separate processors and lower quantities of this parameter will increase the amount of interaction and decrease the diversity difference between islands.

### Migration Percent (migration\_percent)

This optional parameter is for use with the Island method algorithm type. It is used to specify the number of individuals that will migrate from a population as a percentage of the population on a single island. This value should be much less than the total size of the population and usually less than 50%. Larger values will exchange more of the population and may decrease the diversity difference between the islands. Smaller values will exchange less of the population and will cause the islands to act more as independent runs

### Fingerprinting (fingerprinting)

The fingerprinting parameter allows the user to specify whether or not to allow the algorithm to generate structural fingerprints and a fingerprint file that includes the fingerprint distance of each structure from the most fit structure of that generation. In order to use the fingerprinting parameter, the user should also specify the fingerprint bin size (‘fpbin’) and the fingerprint cutoff distance (‘fpcutoff’).

### Fingerprint Bin Size (fpbin)

This parameter is used when applying fingerprinting and will determine the distance bins for sorting the atoms in the structure when constructing the fingerprint function. The default value for this parameter is 0.25 Å. Setting this parameter to smaller sizes will increase the resolution of the fingerprint function and increase its sensitivity to small perturbations in the structure. Decreasing the size of the fingerprint bin will also increase the time needed to calculate it. Generally, for crystal structure optimizations, this parameter should not be similar in size to the minimum expected distance between atoms in the crystal. Higher resolutions are not always necessary as there may be no additional atom in the bin distance and the function would have several empty bins. The user should use the fingerprint file for the minimum energy structure that is output with a fingerprint run in order to see if the bin size is set appropriately for a given structure.

### Fingerprint Cutoff size (fpcutoff)

This parameter is used when applying fingerprinting and is the maximum distance between atoms for the calculation of the fingerprint function. The default value is 15.0 Å. Only atoms within this distance will be counted in the fingerprint function. If this distance exceeds the size of the structure in question then periodic reflections will be applied according to the type of structure being optimized. Setting this parameter too small can cause the function to miss key aspects of the structure and be ineffective for distinguishing different structures. Setting this parameter too large can slow down the optimization and cause unneeded calculations. Viewing the actual fingerprint function that is output in the minimum fingerprint file can be helpful in identifying the proper size of this function. In general, if the fingerprint is being calculated for a crystal structure, the cutoff size should be at minimum the expected length of the longest lattice vector but a good rule of thumb is to try to include at least 2 or 3 unit cells within the cutoff distance. If the fingerprint function is being applied to a defect or cluster simulation, the cutoff distance should be able to incorporate the size of the cluster or region 1 and 2 for a defect. User’s can use the n\_atoms^0.333\*atom\_size rule to estimate the size of the cluster and add this to the twice the size factor (sf) parameter for a defect run. Another good approach is to use a larger then expected fingerprint cutoff for some empirical potential runs to get a feeling for how this size impacts the simulation.

### Fixed Region (fixed\_region)

The fixed region parameter was designed for the optimization of defects in random alloys. The goal of it is to isolate the atoms in region 1 and region 2. If this parameter is set to True, two separate calculator objects are initiated for each structure. One calculator object sets the forces on atoms not included in region 1 and region 2 equal to 0. This allows only those atoms in these regions to determine the structure of the defect during the local relaxation. The second calculator object is then used to calculate the correct energy of the structure. This parameter can only be used with a LAMMPS energy calculation. This is an experimental parameter and should be used with extreme caution. The default value of this parameter is False.

### Constrain Position (constrain\_position)

The constrain position parameter was design for the use in the optimization of defects in random alloys. The goal of this parameter is to restrain the defect within a certain region defined by region 2. The intention is to keep atoms near each other. However, region 2 will adapt but remain close to region 1. The constrain position flag will attempt to keep regions 1 and 2 together. The default value for this parameter is False.

### Rattle Atoms (rattle\_atoms)

The rattle atoms parameter can be set to either true or false. If it is set to true, the atoms in region 1 and region 2 will be randomly moved slightly before each energy evaluation. This parameter can be useful if there are concerns with diversity in the population and producing too many similar structures. However, it is meant more for debugging short restarted runs to improve performance rather than for running an entire optimization. The default value for this parameter is False.

## Population and Individual Generation Parameters

### Size (size)

The size parameter specifies the initial size of the volume that will be used to generate the initial random structures. The default value of the size parameter is equal to the cube root of the number of atoms times the average distance between atoms ([r\_ab](#_R_ab) parameter). The size should be given as a python float value.

For a Defect type simulation, the size can be used to control the location of the interstitials within the bulk crystal. A large size in a Defect simulation will allow the interstitials to be more dispersed within the crystal initially whereas a smaller size will lead to clustering of the interstitials. It is important when evaluating a new problem that this factor is taken into account especially for problems in which the user is unsure whether or not the interstitials will cluster. Additionally, larger sizes initially will make reduce or even provide false positives for convergence criteria. It is recommended that, if the user knows that the interstitials will favor clustering, the size be set small relative to the solid bulk cell size. Note that for the Defect case if the size is larger than the [solid bulk cell size](#_Solid_Cell_Size), atoms will be wrapped into the cell.

It is important to note that the size parameter only sets the initial size of the randomly generated structure. The actual energy relaxation will govern the size of the structure that is optimized.

### Average Atom Distance (r\_ab)

The r\_ab parameter specifies the typical distance between atoms in an individual structure in Angstroms. It has a default value of 2.5 Angstroms. The r\_ab parameter should be specified as a python float value. Typically, the r\_ab parameter is only important for simulations that involve adding or removing atoms as it will allow for better adjustments of the simulation cell size for an individual.

\*Note\*: This parameter does not specify a default distance between atoms. It is primarily used for readjusting simulation cell parameters for multi-atom simulations.

### Custom Generation Types (generate\_flag)

The generate\_flag can be used to determine the method for generating the initial population. The default value for this parameter is ‘box’.

The ‘box’ method will use the size parameter to set the length of each side of a cube. The atoms in the atom list parameter will be randomly generated within this cube.

The ‘sphere’ method will use the size parameter as a diameter for a sphere in which to randomly generate the atoms in the atom list parameter.

The ‘dumbbells’ method is for use when optimizing a defect structure. This method selects atoms that lie within in the bulk structure in the limits of a sphere centered in the structure of the size parameter. It then randomly places the atoms in the atom list parameter near those atoms at random angles to generate random dumbbell configurations.

If ‘random’ is in the generate\_flag parameter (for instance as ‘random\_box’ or ‘random\_sphere’) and a defect structure is being optimized. Then the atoms that fall within region 2 or the region set by a sphere in the center of the bulk material of with a radius given by the size parameter are removed from the structure. The atoms that are in the region are then regenerated to random positions within the region. The defect atoms are then randomly generated according to either the box or sphere configuration as chosen. This creates an amorphous structure within the bulk structure.

### Crystal Specific Parameters

#### Cell shape options (cell\_shape\_options)

The cell shape options parameter is used only in a Crystal type simulation. It allows the user to specify the simulation cell shapes that the algorithm is allowed to explore. With these options set, the algorithm will apply specific constraints to the way in which the cell shape can be altered. This can be useful in cases where the crystal shape is known to be one of only a few options or, in the case of an island method algorithm, to explicitly explore one cell shape per processor. The following cell shapes are available options: cubic, hexagonal, triclinic, monoclinic, orthorhombic, or tetragonal. The default option list includes all of the cell shape options. Alternatively, the user can simply set the cell shape options parameter equal to ‘All’ in order to include all of the possible shapes.

### Defect Specific Parameters

#### Size Factor (sf)

The size factor is used only in Defect type simulations. It is used to specify the amount of surrounding bulk material to include in an individual. The default value is a factor of 1.75. The size factor specifies the radius of a sphere located at the center of the bulk structure.

#### Supercell (supercell)

The supercell parameter is used only in a Defect type simulation. The default value is (2,2,2). The supercell should be structured as a three-valued tuple of integers. It represents the size of the bulk provided in the [solid file](#_Solid_File_(SolidFile)) parameter. It is currently needed only for the CCP-Oct-Tet style crossover.

#### Solid File Name (solidfile)

The solid file parameter is used only in a Defect type simulation. This parameter is **required** for any Defect type simulation. It specifies the filename for the bulk structure to be used. It should be input as a python string using the full filename and extension. The solid file must be formatted as an ‘.xyz’ atom file.

#### Solid Cell Size (solidcell)

The solid cell parameter is used only in a Defect type simulation. It specifies the simulation cell size for structure provided in the [solid file](#_Solid_File_(SolidFile)). Use of this parameter is highly recommended for any Defect type simulation. The solid cell should be specified as a three-valued tuple specifying the size in the (x,y,z) directions. It can also be a numpy array of dimensions 3x3 as specified by the ASE manual. If the solid cell size is not specified, then it will be assumed to be equal to the distance between the first and third atom listed in the solid file.

#### Evaluation of Solid (evalsolid)

The solid evaluation parameter is used only in a Defect type simulation. It can be set to either true/false. The default value is false. A value of true will allow the energy evaluation code (Molecular Dynamics LAMMPS or DFT VASP) to use the parameters provided to calculate the energy of the bulk solid. This information will be output to the Summary file. It is useful as a reference point or for formation energy calculations. It is convenient to set this parameter to false if the provided potential is not able to calculate the energy of the bulk as well as the defect bulk structure, as is common in simulations with impurities.

#### Find defect scheme (finddefects)

The find defect scheme parameter is used only in a Defect type simulation. It can be set to either true/false. The default value is false. For more effective convergence, this parameter should be set to true. This scheme allows the position and type of any vacancy or interstitial to be identified and isolated to the individual after the local structural optimization. If the parameter is set to False, then the structure output by the energy evaluation code will simply be split so that the first few atoms are assigned to the individual and the last few atoms are assigned to the bulk structure.

#### Track Vacancies (trackvacs)

The track vacancies flag option is used for defect structure optimizations. The default value is false, however, if the value is set to true then structural vacancies will be tracked through the optimization and they will be output to the structures as Xx atoms. Note that the vacancy structure optimization is still in testing.

#### Track Swaps (trackswaps)

The track swaps flag option is used to track atoms in a structure that get exchanged between the bulk and defect/interstitial positions. It will particularly track changes in elements in the bulk structure and mark them as defects. This will ensure that these atoms are included in region 1 and region 2.

#### Alloy (alloy)

The alloy parameter is used only in a Defect type simulation. The alloy parameter allows for the atoms in the individual to interact with the atoms in the bulk structure. This parameter may be set to either true or false. It is recommended that for almost all cases, this parameter should be set to true. The default value of the alloy parameter is true. Setting this parameter to false will greatly increase the computational time required.

#### Random vacancy location start (random\_vac\_start)

The random vacancy start flag can be set to either true or false. If the flag is true, then vacancy atoms can be selected from anywhere in the bulk structure rather than just near the center of the structure. The default value for this parameter is False.

#### Random Location Start (random\_loc\_start)

The random location starting parameter is used only for the Defect case. This parameter may be set to either true or false. The default setting is false. This parameter controls the starting position of the interstitials within the bulk structure or rather the position of the center of a box of size equal to the value of the [size](#_Size) parameter. If this parameter is set to False, then the interstitials will be placed about the center of the structure provided in the [solid file](#_Solid_File_Name). If the random location start option is activated by setting it equal to true, then each structure will have a different starting location within the [solid file](#_Solid_Cell_Size) boundary cell. Atoms will be wrapped into the solid cell. Users should be cautious when using this parameter, as it will cause difficulty in creating successful crossovers when large [size factors](#_Size_Factor_(sf)) are involved. This is because the crossovers will be attempting to combine structures from different locations in the bulk solid. Setting the size factor equal to zero when using this feature can eliminate this problem.

## Evaluation Parameters

### Calculation Method (calc\_method)

The calculation method parameter allows the user to specify the desired method for the calculation of the energy of the individual structures. Current available options for this parameter include: ‘VASP’, ‘LAMMPS’, ‘MAST\_LAMMPS’, ‘MAST\_VASP’. If the vaspcalc parameter is not specified, then the default value for this parameter is ‘LAMMPS’, otherwise the default is ‘VASP’. The ‘MAST’ methods are used in conjunction with running the program with Mast and will prevent the program from calculating the energy internally.

### VASP calculator (vaspcalc)

The VASP calculator parameter allows the user to specify an ASE calculator class VASP object. This provides the details for how the simulation should conduct the VASP calculations. Please refer to the ASE documentation for how to specify specific inputs for the VASP calculator (<https://wiki.fysik.dtu.dk/ase/ase/calculators/vasp.html>).

### Pair Style (pair\_style)

The pair style parameter allows the user to specify the pair style desired for the LAMMPS calculation. Available options include the following: Although, it is important to note that theoretically any pair style available in LAMMPS can be used with the StructOpt code. The default pair style will assume a Leonard-Jones potential. Available options for pair style include: ‘tersoff’, ‘eam’, ‘eam/fs’, ‘eam/cd’, ‘edip’, ‘bop’, ‘buck’, and ‘other’. Setting this parameter equal to ‘other’ signals StructOpt to use the parameters for other potentials.

### BOP Cutoff (bopcutoff)

The BOP cutoff parameter is for use exclusively with the LAMMPS BOP pair style. If a BOP pair style is used, then the BOP cutoff must be specified or the program will error. See the LAMMPS documentation for details on the use of a cutoff parameter with the BOP potential.

### Buckingham cutoff (buckcutoff)

The Buckingham cutoff parameter is for use exclusively with the LAMMPS Buckingham potential. The default value for this parameter is 1. See the LAMMPS documentation for how to specify the Buckingham cutoff parameter.

### Buckingham parameters (buckparameters)

The Buckingham parameters input is for use exclusively with the LAMMPS Buckingham potential. The default value is: [‘\* \* 100.0 1.5 200.0’]. See the LAMMPS documentation for how to specify the Buckingham parameters.

### Other Pair Style (ps\_other)

The ps\_other parameter is used when the pair style has been set to other. It is used to specify other parameters for use in the energy evaluation with LAMMPS. These parameters include specifying newton and charges for the structure. The default value is ‘None’.

### Pair Style Name (ps\_name)

The pair style name parameter is for use exclusively with the ‘other’ selection of the pair style parameter. The pair style name communicates to LAMMPS what alternative pair style to use for the evaluation. This parameter is useful when experimenting with a compound potential or with a potential that has not yet been added to the StructOpt code. The default value is None.

### Pair Coefficients (pair\_coeff)

The pair coefficients parameter is for use exclusively with the ‘other’ selection of the pair style parameter. The pair coefficients must be provided in order to use the ‘other’ pair style parameter. Otherwise, the code will generate an error.

### Potential Files (pot\_file)

The potential files parameter is for use with any LAMMPS potential that requires use of a dedicated file. Multiple files can be specified. The default value is None. This parameter should be set equal to the filename.

### Keep LAMMPS Files (lammps\_keep\_files)

The keep LAMMPS files parameter is for use with the LAMMPS or mixed calculation method. It may be set to either true or false. The default value is false. This parameter determines whether or not to keep the LAMMPS files for each structural evaluation. Note that setting this parameter equal to true will result in the generation of many files even for modest runs. However, this can be useful when attempting to debug a LAMMPS potential. These files will be written to a folder within the output folder by the name of ‘LAMMPSFiles’. It should also be noted that for multi-node jobs this parameter should be set to True.

### LAMMPS Minimization (lammps\_min)

The LAMMPS minimization parameter is for use with the LAMMPS calculation method. It provides the minimization command for the LAMMPS local minimizer. See the LAMMPS documentation for more information about how to specify the LAMMPS minimize command. The default value for this parameter is None. This will result in no local minimization of the structures during the simulation and will greatly increase the computation time requirements. Users can also use this parameter to set up a local quench rather than a simple local minimization.

### LAMMPS Minimizer Style (lammps\_min\_style)

The LAMMPS minimizer style is for use with the LAMMPS local minimizer. This parameter allows users to specify the type of local optimizer to be used in the simulation. See the LAMMPS documentation for available local minimizer options. The default value of this parameter is ‘cg’. This will activate the conjugate gradient local minimizer.

### Number of LAMMPS thermo steps (lammps\_thermo\_steps)

The number of LAMMPS thermo steps parameter is for use with the LAMMPS calculator minimizer. It specifies how frequently the LAMMPS output files should be updates. The default value for this parameter is 1. This will output the atom position, force, and velocity data every step of the local minimization. Setting this parameter to larger values can help to control the file size of the LAMMPS output and is particularly important for optimizations of structures with thousands of atoms or with long local optimizations or quenching procedures. It may be necessary to specify a larger value for this parameter if the quench mutation or minimizer is used.

### ASE Minimizer (ase\_min)

The ASE minimizer is an alternative option to the LAMMPS or VASP minimizer. Refer to the ASE documentation for information on these local minimizers (<https://wiki.fysik.dtu.dk/ase/ase/optimize.html?highlight=minimizer>). Note that this minimizer will take longer than the LAMMPS minimizer as it will require more communication between python and LAMMPS. It is therefore not suggested. However, for testing a simple calculation or for use with the ASE Lennard Jones optimizer, it can be useful. It should be noted that there is an issue with file input and output in connection with the ASE minimizer and the algorithm can often produce errors such as “Too many open files” if run for several generations or large populations.

### Maximum Solver Force (ase\_min\_fmax)

The maximum solver force parameter is for use with the ASE minimizer option. It specifies the maximum tolerable force between atoms for convergence. The default value is 0.01.

### Maximum Solver Steps (ase\_min\_maxsteps)

The maximum solver steps parameter is for use with the ASE minimizer option. It specifies the maximum number of tolerable steps for the minimizer before convergence is reached. The default value is 2500.

### Size of Large Box (large\_box\_size)

This parameter sets the size of the periodic box that the non-periodic cluster structure will be embedded in for an energy calculation. The default size for this parameter is 500 Å. This will center the cluster in a cubic box with each side length equal to 500 Å.

### Energy Cutoff Factor (energy\_cutoff\_factor)

The energy cutoff factor is a cutoff value for the most energy per atom. It is used to identify if a energy calculation has been performed incorrectly. The default value for this parameter is 10.0. This means that, if the absolute value of the energy calculated for a five atom system exceeds 50 eV, the algorithm will assume that something went wrong with the energy calculation and will set the energy and fitness of that structure to 1000 in order to attempt to remove it from the population.

## Crossover Parameters

### Crossover Probability (cxpb)

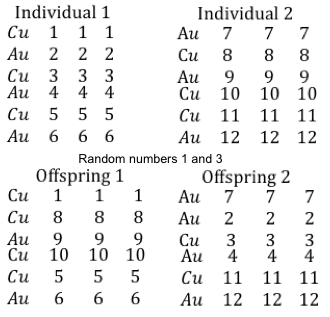
The crossover probability parameter determines the probability that a pair of parent structures will mate and produce an offspring. The default value for this parameter is 0.8, or structures have an 80% likelihood of mating and producing an offspring. This value should be between 0.0 and 1.0. A value of 0.0 will result in no crossovers throughout the simulation. A value of 1.0 means every structure pair will mate and produce an offspring. This will result in nindiv offspring. Setting this parameter equal to 1.0 means that the mutants produced in the population will only be mutants of the offspring and no mutants of the parent structures will be produced. This can reduce the ability of the program to identify higher fitness structures in later generations.

### Crossover Scheme (cx\_scheme)

The crossover scheme parameter allows the user to select the preferred method of mating or exchanging two structures. The default value for this parameter is ‘cxtp’.

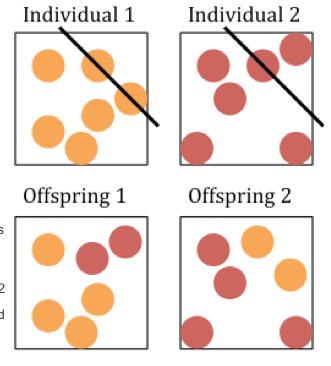
#### Two-Point Crossover (cxtp, cxtpa, cxtpc)

The two-point crossover is the most basic method for exchanging atoms between two structures. It treats atom positions as simple matrix of numbers. The crossover proceeds by randomly selecting two numbers between 0 and the max number of atoms, inclusively selecting atoms between the two random numbers in each individual, and swapping the atom group between each individual. This crossover will always maintain the total number of atoms in each individual. The ‘cxtpa’ version allows for concentration control. The ‘cxtpc’ version uses coordinates as fraction of the cell rather than real coordinates to assist in crystal structure optimizations. See the figure below for an example of how this crossover will proceed.



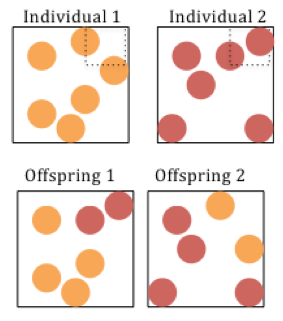
#### Cut and Splice crossover (rotct, rotct\_rand, rotct\_defect, rotct\_rand\_clus, rotct\_rand\_defect)

The cut and splice crossover involves selecting a plane through each structure and exchanging the atoms on either side of that plane. It was first proposed by Deaven and Ho. The main difference between the two options is that the random plane will always be selected through the center of mass in the ‘rotct’ version. The crossover allows for optionally maintaining the concentration of atoms. For concentration control, if the total number of atoms in offspring 1 is less than the total number of atoms in individual 1 then the atoms in offspring 2 that are closest in distance to the random plane are removed from offspring 2 and added to offspring 1.



#### Box Cut Crossover (clustbx, randalloybox, newclus)

The box cut crossover is targeted for ‘Defect’ structure simulations. It creates a small box within the structure that can be reflected over periodic boundaries and exchanges the atoms between structures that fall within this box.



## Mutation Parameters

### Mutation Probability (mutpb)

The mutation probability parameter specifies the likelihood of a structure undergoing a mutation. The default value for this parameter is 0.0. This means that no individuals will undergo a mutation. In general, recommended values for the mutation probability fall within 0.10 and 0.25 meaning between 10% and 25% of the individuals in the population will mutate. This helps to ensure the population does not get caught in a potential energy funnel.

### Mutant Add (mutant\_add)

The mutant add flag can be set to either True or False. The default value is False. If the value is set to True, then mutants will be added to the offspring as separate structures rather than replace structures in the offspring. For instance, if a crossover results in a combination of individual 1 and 2 and then this resulting structure is mutated, the mutant add structure will allow both the individual 1+2 structure and the individual 1+2 mutated structure to be evaluated rather than just the 1+2 mutated structure. Depending on the mutation probability, turning this flag on will result in a much larger number of evaluations per generation. This flag may be useful if mutations are generally less successful than crossovers (as is usually the case early on in the evolution of the structure) as it will allow the crossovers to be evaluated independently.

### Mutation Options (mutation\_options)

The mutation options parameter specifies which mutations are allowed to occur during the simulation. It is input as a list of available options. The optimizer will randomly choose an option from this list and apply that mutation. In order to increase the probability that the optimizer will choose a particular mutation type, the user can simply list that mutation multiple times. Once a mutation is applied, the structure will receive a history key in the history index indicating the type of mutation and some information about the mutation such as the number of atoms involved or the number of steps used.

#### ASE Minimization Mutation (ase\_minimization)

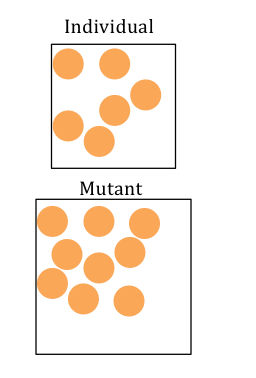
The ASE minimization mutation uses the built in ASE BFGS minimizer to locally relax the positions of the atoms in the structure. This mutation relies on several static energy calculations. Users should be cautious about using this mutation for large systems or if using VASP for the energy calculations without MAST. This mutation will use a maximum number of 2500 steps and attempt to resolve the structure to a maximum force value of 0.01. The mutation history key for this mutation is ‘ASEM’.

#### Add/Remove Atoms Mutation (atoms\_add, atoms\_remove)

The add or remove atoms mutation allows for addition or subtraction of atoms in an individual. This mutation will scale the size of the cell according to the number of atoms added or removed according to the equation:

natoms1/3\* r\_ab

The history key for this mutation is ‘AAX’ or ‘RAX’ for atoms\_add and atoms\_remove respectively where ‘X’ represents the number of atoms added or removed.



#### Basin Hopping Mutations (basin\_hop\_la, basin\_hop\_permute, basin\_hop\_rattle, basin\_hop\_rotate)

The basin hopping mutations allow for small basin hopping optimizations that apply lattice alteration, permutation, rattle, or rotation mutations. Please refer to the specific mutations for information on how they work. The basin hopping algorithm will attempt to identify a lower energy structure than the starting structure. The history key for these mutations are ‘BHLAX’, ‘BHPX’, ‘BHARX’, ‘BHRatX’, and ‘BHRX’ where X is the number of steps taken to identify the structure.

##### Basin Hopping Temperature (bh\_temp)

This parameter is used for basin hopping runs and sets the temperature for the Metropolis criterion. This temperature should be given in kT. The default value for this parameter is 0.0862 or 1000K.

##### Basin Hopping Steps (bh\_steps)

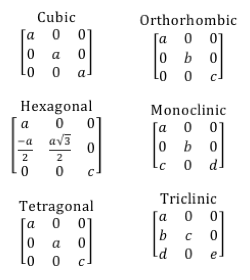
This parameter is used for basin hopping runs and sets the maximum number of steps to use to identify a lower energy structure. The default value for this parameter is 100 steps.

#### Cell Relaxation Mutation (cell\_relax\_lammps)

The cell relaxation mutation uses the LAMMPS box/relax command to relax the lattice parameters of the structure. This relaxation assumes isometric pressure of 0.0 bars and a max volume change for an iteration of 0.001. Refer to the LAMMPS documentation for more information on exactly how this relaxation method works. The history key for this mutation is ‘LBR’.

#### Cell Shape Mutation (cell\_shape)

The cell shape mutation is targeted at ‘Crystal’ type simulations. It allows for the alteration of the individuals computational cell by specifically altering the lattice parameters to fall into one of the six identified structures. The history key for this mutation is ‘CSX’ where ‘X’ represents the type of cell the structure has been transformed into. The ‘X’ value can be ‘C’ for a change to a cubic cell, ‘H’ for a hexagonal cell, ‘Te’ for a tetragonal cell, ‘O’ for an orthorhombic cell, ‘M’ for a monoclinic cell, and ‘Tr’ for a triclinic cell.



#### Lattice Alteration Mutations (lattice\_alteration, lattice\_alteration\_crystal, lattice\_alteration\_group, lattice\_alteration\_nn, lattice\_alteration\_rdrd, lattice\_alteration\_small)

The lattice alteration mutations allow for the movement of an atom or group of atoms within the structure to new positions.

The basic lattice alteration mutation will select random atoms and move them to a randomly generated point within the bounds of the maximum and minimum positions of the atoms in the structure. For a defect structure, the positions of the atoms in set one and two limit these bounds. It will apply these random moves to at most 1/5 of the atoms in the structure.

The crystal implementation of this mutation will select random atoms and move them to a randomly generated point within the bounds of the structure cell. It will apply these random moves to at most 1/5 of the atoms in the structure.

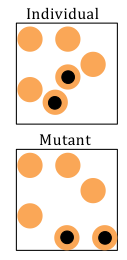
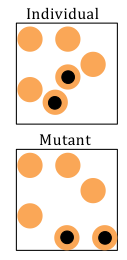
The group implementation of this mutation will select atoms closest to a random point and move them collectively to a point within the bounds of the maximum and minimum positions of the atoms in the structure. For a defect structure, the positions of the atoms in set one and two limit these bounds. It will move at most 1/5 of the atoms in the structure.

The ‘nn’ or nearest neighbor implementation of this mutation moves an atom by the nearest neighbor distance in the positive or negative x, y, or z direction. This mutation will move at most half of the atoms in the structure. It is primarily designed for use with cubic systems. Users should be careful using this mutation in combination with large structures, as this algorithm will create several nearest neighbor lists that may become unreasonable for larger numbers.

The ‘rdrd’ or random distance random direction implementation of this mutation moves at most 1/5 of the atoms in the structure by a distance between 0.5 Å and the maximum length of the structure. It has access to a full 360 degrees of random direction from which to choose. This implementation is the most robust and is suggested for user’s doing a cursory exploration of a given system.

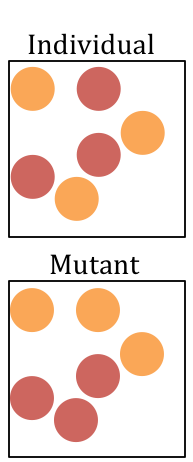
The small implementation of this mutation will select atoms in the system and move them at most one angstrom from their starting distance. This mutation will move at most 1/5 of the atoms in the structure.

The history keys for this mutation are ‘LAX’, ‘LACX’, ‘LAGCX’, ‘LANNX’, ‘LARDX’, and ‘LASX’ respectively where ‘X’ is the number of atoms that are moved.



#### Permutation Mutations (permutation, permutation\_crystal, permutation\_crystal\_multi, permutation\_bulk)

The permutation mutation allows for the type of atom at a given position to be exchanged within a structure. Only one atom is exchanged for a single mutation. The permutation crystal method is particularly for optimizing defect structures. It allows for atom types to be exchanged between set one and two and set three. The permutation crystal multiple method applies the permutation to multiple atoms in a single mutation. The permutation bulk option is specifically for the optimization of a defect structure and exchanges atoms only in set three. The history keys for this mutation are ‘P’, ‘PC’, ‘PCMAX’ (where ‘X’ is the number of atoms permuted), and ‘PB’.



#### Quench Mutation (quench)

The quench mutation is used to apply a molecular dynamics thermal quench to the system. This mutation is set up to work only with LAMMPS. In this mutation, there are two quenches. The first quench is usually a rapid cooling and the second is a slower cooling. Between each quench relaxation there is a local conjugate gradient relaxation. The history key for this mutation is ‘Q’. The parameters for specifying the details of the quench relaxation are given below.

##### Quench\_max\_temp

The max temp parameter is used for the quench mutation to determine the highest temp to be applied to the system. This temperature should be given in Kelvin. The default value for this parameter is 1000K.

##### Quench\_min\_temp

The min temp parameter is used for the quench mutation to determine the temperature to cool the system to. This temperature should be given in Kelvin. This temperature must be greater than 0K due to limitations in LAMMPS. The default value for this parameter is 2K.

##### Quench\_step\_size

The quench step size is used for the quench mutation and is the parameter provided to the LAMMPS nvt minimization as the tdamp parameter. This is given in picoseconds. The default value for this parameter is 0.01ps.

##### Quench\_n\_steps\_1

The number of steps used for the initial quick quench is given by the quench\_n\_steps\_1 parameter. This is the number of time steps to be used by for the LAMMPS nvt relaxation. The default value for this parameter is 10,000 steps.

##### Quench\_number\_steps\_2

The number of steps used for the longer quench is given by the quench\_n\_steps\_2 parameter. This is the number of time steps to be used by for the LAMMPS nvt relaxation. The default value for this parameter is twice the number of steps used in the initial quick quench relaxation as given by the quench\_n\_steps\_1 parameter.

#### Random Replacement Mutation (random\_replacement)

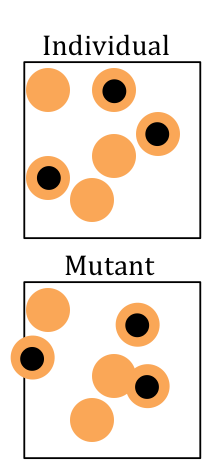
The random replacement mutation replaces a selection of atoms in the structure with a new randomly generated set of atoms. The mutation replaces at most ½ of the atoms in the entire structure. The atoms that are selected for replacement are selected based on geometric proximity. The history key for this mutation is ‘RGRX’ where ‘X’ is the number of atoms that were replaced by the mutation.

#### Rattle Mutation

The rattle mutation applies a Gaussian movement to all of the atoms in the structure. The history key for this mutation is ‘Rat’. It is most useful for the optimization of defect structures.

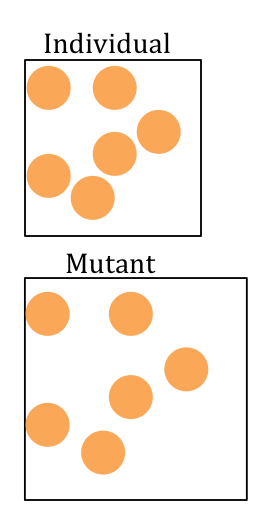
#### Rotation Mutations (rotation, rotation\_geo)

The rotation mutation allows for the rotation of a group of atoms within the structure. The standard rotation version rotates a random selection of atoms while the geometric version selects a group of nearby atoms. The rotation occurs around the center of mass of the selected group. At most ½ of the atoms in the structure will be rotated in a single mutation. The history key for this mutation is ‘RX’ and ‘GRX’ where ‘X’ is the number of atoms rotated in the mutation.



#### Scale Size Mutation (Scale\_size)

The size scaling mutation allows for the alteration of the cell size in a ‘Crystal’ type simulation. It can scale up or down the size of the cell. The history key for this mutation is ‘SSz’.



#### Swap Mutations (swap, swap\_int, swap\_int\_local, swap\_vacancy)

The swap mutation is similar to the permutation mutation in that it alters the type of atom at a given position with the exception that the atom need not maintain concentration elsewhere. This mutation is particularly designed for the optimization of defect structures that have variable composition.

#### Zero Point Rotation Mutations (zp\_rotation, zp\_rotation\_fixed)

The zero point rotation mutations are designed for use with cluster optimizations. It rotates all of the atoms in the structure. Structures mutated with this mutation should not change energy as the atoms are rotated as a whole. The fixed implementation of this mutation rotates the structure by 30, 45, 60, 90, 120, 135, 150, 180.

## Selection Parameters

### Selection Scheme (selection\_scheme)

The selection scheme parameter sets the method for determining which structures will mate. The selection scheme chooses how to generate a list of selected structures from a population. The order in which the structures are selected will control which structures will mate. The default value for this parameter is ‘tournament2’ selection.

### Natural Selection Scheme (natural\_selection\_scheme)

The natural selection scheme allows for the selection of the method for determining which individuals in a population are allowed to survive to the next generation. It utilizes the same options as available to the selection scheme parameter. The default value for the natural selection scheme is ‘’metropolis’ for simulated annealing and basin hopping runs, ‘tournament’ for genetic algorithm optimizations, and ‘best’ for random optimizations or when the optimization type is not specified.

#### Best (best)

The best selection orders the population from most fit to least fit to generate the selection list. If this is used for a selection scheme, the most fit will only undergo a crossover with the second most fit, the third most fit will be paired with the fourth most fit, and so on. This method will not allow for any structures to be repeated more than once.

#### Cost (cost)

The cost selection option is a roulette wheel selection method. Each individual is given a selection probability relative to the least fit individual in the population. This method will allow duplicate structures to exist within the selection list but will prevent duplicates from being selected adjacently, which will prevent a structure from attempting a crossover with itself. The primary difference between the cost selection method and the rank selection method is the weighting each individual receives. In the cost selection method, the weight is assigned by not only giving those with higher fitness a higher weight but by giving the higher fitness individuals a proportionally higher weight. This can be useful in populations with higher diversity, as it will proportionally favor those individuals with higher fitness better than the rank selection method.

#### Fixed Uniform Selection Scheme (fuss, fuss1, fuss1r, fuss2, fussf, fussr)

The fixed uniform selection method is targeted at maintaining a high diversity population even at the expense of convergence. In this method, a random number is chosen between the lowest fitness and highest fitness. The structure with a fitness closest to that random point is then added to the selection list. In order to reduce possible divergence, a limit on the allowable range for this random value can be specified using the ‘fusslimit’ parameter. The default value for this parameter is 10. This means that if the difference between the lowest fitness and highest fitness in the population is greater than 10 then the random point will be chosen within the range of the highest fitness and the highest fitness minus 10. The ‘FUSS’ option of this selection method will select a single random point limited by the ‘fusslimit’ parameter and populate the selection list based on the proximity of a structure’s fitness to that point. It will always prevent any duplicates from entering the selection list and will ensure that the best fitness individual survives the selection. The alternate ‘FUSS1’ option will select a new point from the entire fitness range without any limits irrespective of the ‘fusslimit’ parameter for each individual in the population list. It will also ensure the survival of the best fitness individual. The ‘FUSS2’ option allows for the selection of a new point for each individual within the bounds of the ‘fusslimit’ parameter. It will also ensure the survival of the best fitness individual. Finally, the ‘FUSSF’ option of this selection method will use a structure’s fingerprint distance to calculate the distance to a single random selection point. It will also ensure the survival of the best fitness individual.

#### Metropolis (metropolis)

The metropolis selection method selects structures from the population based on the Metropolis criterion.

##### Metropolis Temperature (metropolis\_temp)

The metropolis temperature parameter sets the temperature for the Metropolis selection criterion for the metropolis selection method. The default value for this parameter is 30.0.

#### Random Selection (random\_pick)

The random selection method randomly chooses structures from the population to generate the selection list. This method allows for structures to be repeated more than once as well as for structures to be selected adjacently. This would allow for a structure to attempt a crossover with itself if this method is used for the selection scheme.

#### Rank (rank)

The rank selection option is a roulette wheel selection method. Each individual is given a selection probability based on its ranking when listed from most fit to least fit. This method will allow duplicate structures to exist within the selection list but will prevent duplicates from being selected adjacently, which will prevent a structure from attempting a crossover with itself. Often this results in the highest ranked individual being paired with multiple lower fitness individuals.

#### Tournament (tournament, tournament1, tournament2)

The tournament selection method allows for competition within subpopulations for selection preferences. In this method, the subpopulations are randomly generated from the total population. The size of the subpopulation is set using the ‘tournsize’ parameter. The default value for this parameter is three. The most fit individual from this subpopulation is then added to the selection list. The ‘tournament’ option of this selection method will prevent duplicate structures from appearing in the selection list and will ensure that the most fit individual in the population survives the selection process. The alternate ‘tournament1’ option of this selection method will allow for duplicate structures to appear in the selection list as well as appear adjacently. The alternate ‘tournament2’ option of this selection method will allow for duplicate structures within the population list as long as they do not appear adjacently.

### Fitness Scheme (FIT\_SCHEME)

The fitness scheme parameter allows the user to select the preferred method for calculating the fitness of an individual.

#### Chemical Potential Swap Energy (chempotswap)

The chemical potential swap energy scheme is for use with a swap mutation in a ‘Defect’ type simulation. It takes into account the energy of the atoms in an external box that have been removed from the structure. This fitness scheme is designed for systems with varying concentration.

#### Energy per Atom (enpafit)

This scheme divides the total potential energy calculated for a structure by the number of atoms in the structure in order to use the energy per atom of the calculated structure as the fitness. This fitness scheme can be useful if the number of atoms in the system will vary.

#### Enthalpy (enthalpyfit)

This scheme calculates the enthalpy of the structure for the fitness according to the formula:

H = E + PV

It uses the potential energy (E) calculated by the local relaxation as well as the volume (V) and pressure (P) from the calculation. This fitness scheme is suggested for use with crystal structure optimizations.

#### Exponential (exponentialfit)

This scheme uses an exponential function to weight the differences between the total energies of the calculated structure as the fitness according to the following equation:

The minimum and maximum energy represent the minimum and maximum energy from the previous population. The idea behind this fitness scheme is to allow structures to be weighted differently as the population begins to converge. It is best to use this fitness scheme in conjunction with the ‘cost’ or ‘rank’ selection schemes.

#### Formation Energy (formationenergy)

The formation energy scheme calculates the formation energy of the structure for use as the fitness according to the following equation:

Where the chemical potential used here is the one given in the atomlist parameter and the summation acts over all of the atom types in the system.

#### Silicon Bias (sibias)

The silicon bias scheme is a system specific fitness scheme that identifies the composition of the interstitial defects and adds 50 eV per interstitial Si atom to the energy of the structure for the fitness. The goal of this fitness scheme is to bias the system away from structures that produce Si atoms as interstitials.

#### STEM Cost Function (stem\_cost, stem\_cost\_rotation)

The STEM cost function uses the total energy of the structure and the error between a simulated image and experimental image as the fitness. The rotation implementation of this method uses a rotation to try to minimize the error between the two images.

#### Total energy (totalenfit)

This scheme uses the total energy of the calculated structure as the fitness. For most optimization cases, this is the suggested fitness scheme.

### Convergence Scheme (convergence\_scheme)

The convergence scheme parameter allows the user to specify the method for determining the convergence of a simulation and the ending conditions. The default value for this parameter is ‘max\_gen’.

##### maxgen

This parameter specifies the maximum number of generations that the simulation will run regardless of if the other convergence criteria have been met. The default value for this parameter is 5.

##### reqrep

The required repetitions (reqrep) parameter specifies the number of generations a low fitness structure should be repeated before the convergence is satisfied.

##### tolerance

The tolerance parameter specifies the maximum difference between the fitness of two structures for them to be considered equivalent. It is used to determine if the minimum fitness structure is the same as the previous generation in order to determine the convergence.

#### Maximum Generations (max\_gen)

The maximum generation option for convergence terminates the simulation after a specified number of generations regardless of convergence. The maximum number of generations is specified using the ‘maxgen’ parameter. The default value for this parameter is 5 generations. Recommended values for the max\_gen parameter range from 50 to 500 depending on the simulation.

#### Minimum Generations of Stable Fitness (gen\_rep\_min)

This option for convergence specifies that if the lowest fitness in a population has been stable for a minimum of generations then the simulation can be terminated. The required number of generations of repeated stability is set using the ‘reqrep’ parameter. The default value of this parameter is 10. Typical values for this parameter can fall between 10 and 200 depending on the simulation. Additionally, the ‘tolerance’ parameter specifies the maximum difference between two fitnesses for them to be considered equivalent. The default value for this parameter is 0.001.

#### Minimum Generations of Average Fitness (gen\_rep\_avg)

This option for convergence specifies that if the average fitness of a population has been stable for a minimum number of generations, then the simulation is terminated. The required number of generations of repeated stability is set using the ‘reqrep’ parameter. The default value of this parameter is 10. Typical values for this parameter can fall between 10 and 200 depending on the simulation. Additionally, the ‘tolerance’ parameter specifies the maximum difference between two fitnesses for them to be considered equivalent. The default value for this parameter is 0.001.

#### Fitness Standard Deviation (std)

This option for convergence specifies that if the standard deviation of the fitness of the population falls beneath a certain value then the simulation is terminated.

### Convergence Control Scheme (predator)

The convergence control scheme is used to maintain the diversity of a population. For some simulations, especially ‘Defect’ type simulations, the algorithm can often produce duplicate structures from different starting points. To reduce the likelihood of a population filling with equivalent structures, a convergence control scheme or predator scheme is used. The default value of this parameter is ‘fitpred’, which assumes a minimum difference between fitnesses to determine if structures are similar. The minimum difference value is set using the ‘demin’ parameter. The default value for this parameter is 0.005.

### demin

The demin parameter specifies the minimum difference in fitness between two structures for them to be considered identical for the predator to remove one from the population. The default value for this parameter is 0.005

#### Adapting Predator (adapting)

This predator scheme is meant to adapt to the population as it evolves. It does this by using the fixed uniform selection scheme for the natural selection method but basing the fusslimit on the number of generations.

### adaptbegin

The adaptbegin parameter is the fraction of the total number of required repetitions (reqrep) or max generations before the adapting predator will start to take effect. The default value for this parameter is 0.75.

### adaptmultiplier

The adaptmultiplier parameter governs how much the fusslimit will change as the adapting predator is applied. The default value for this parameter is 3.0.

#### Energy Cluster (energy\_cluster)

The energy cluster predator uses a 1D k-means clustering algorithm to separate the population into nindiv groups of structures that are energetically similar. The algorithm will choose one structure from each of these groups to survive to the next generation.

#### Fingerprint Niche Predator (fingerprint\_niche)

This predator uses a 2D k-means clustering algorithm to separate the population into nindiv groups of structures. Grouping fingerprint distance and energy identifies the clusters of structures and one structure from each group is selected for survival. If the k-means cluster algorithm fails to identify clusters then the ‘mutation\_dups’ predator is used instead.

#### Fitness Predator (fitpred, fitpred\_bests, fitpred\_new)

This predator scheme uses the fitness of the structures in the population to eliminate duplicates. The bests implementation of this predator replaces any deficits in the size of the population with structures from the list of best structures. The new implementation of this predator replaces any deficits in the size of the population with a randomly generated new structure.

#### Mutation Duplicate Predator (mutation\_dups, mutation\_dups\_energy, mutation\_dups\_quench, mutation\_dups\_zp)

The mutation duplicate predator is the most commonly applied predator. It removes any duplicate structures and applies the chosen natural selection scheme. If there are too few structures in the population after the predator, then the deficit is made up by mutating some of the duplicated structures. The energy implementation of this predator method compares the energy of the structures in order to establish if two structures are duplicates rather than the fitness. The quench implementation of this predator uses only a slow quench mutation to alter the duplicate structure in order to fill any deficits in the number of structures in the population. The zp implementation of this predator method applies only the zero point rotation mutation to alter any duplicate structures that needed to be added to the population.

# 

# Reference Listing

This section provides an alphabetical list of functions with clear concise description of inputs and outputs.

adaptbegin – Adapting Fitness function generation to initiate. Default value: 0.75\*genrep

adaptmulitplier – Adapting Fitness function quantity to reduce. Default value: 3.0

algorithm\_type – Select type of algorithm. Options: ‘lambda+mu’, ‘Island\_Method’, ‘lambda,mu’, ‘Island\_Method\_lambda,mu’

allenergyfile – Determine if to output all energy file for simulation. Default value: False

alloy - \*\*Degenerate\*\* Always set to: True

ase\_min – Determine if to use built in ASE minimizers. Default value: False

ase\_min\_fmax – Maximum value of force for ASE structure minimizer. Default value: 0.01

ase\_min\_maxsteps – Maximum number of steps to complete for ASE structure minimizer. Default value: 2500

atomlist – Sets the number and type of atoms in a simulation. Structure as: [(symbol, concentration, mass, potential)]

best\_inds\_list – Determine if to output and track best structures. Default value: True

bh\_steps – Number of steps to run in Basin hopping mutation. Default value: 100

bh\_temp – Temperature for basin hopping acceptance given in (kT). Default value: 0.086

bopcutoff – Cutoff distance for BOP potential

buckcutoff – Cutoff distance for Buckingham potential. Default value: 1

buckparameters – Parameters for Buckingham potential. Default value: ['\* \* 100.00 1.5 200.0']

calc\_method – Calculation method. Available options: VASP, LAMMPS, MAST\_VASP, MAST\_LAMMPS, LennardJones

cell\_shape\_options – Options for crystal structure optimizer to explore. Should be given as a list of strings. Available options: ['cubic', 'hexagonal', 'triclinic', 'monoclinic', 'orthorhombic', 'tetragonal’]

constrain\_position – Option to constrain the area in which a defect can be located. Default value: False.

constrain\_swaps – Option to set an external value of available atoms for potential. Default value: None.

convergence\_scheme – Scheme to determine when the algorithm has converged. Default value: max\_gen. Available options: max\_gen, gen\_rep\_min, gen\_rep\_avg, std.

cx\_scheme – Scheme to be used for crossover. Default value: 'cxtp'. Available options: clustbx, cxtp, cxtpa, cxtpc, newclus, randalloybox, rotct\_rand\_clus, rotct\_rand, rotct

cxpb - Crossover Probability. Default value for GA: 0.8. Default value for other: 0.0

debug – Current method for debugging a section. Should be provided as a list of strings Default value is no debugging: ['None']. Available options: [‘None’, ‘MA’, ‘SEL’, ‘MU’, ‘CX’, ‘FIT’, ‘EE’, ‘FD’, ‘SetCalc’]

demin – Difference in fitness for considering a structure identical to another. Default value: 0.005

energy\_cutoff\_factor – Value for the energy per atom limit used to identify problems with energy calculations. Default value: 10.0

evalsolid – Flag for determining if a bulk solid should be evaluated for a given system. Default value: False

filename – Name of output folder and file for optimizer data. Default value: ‘Output'

finddefects – Flag for determining if the defects should be tracked through the evolution or assumed constant. Default value: True

fingerprinting – Flag for determining if fingerprint function for structures should be calculated. Default value: False

fitness\_scheme – Method for determining the fitness of an individual. Default value: 'totalenfit'. Available options: chempotswap, enpafit, enthalpyfit, exponential fit, formationenergy, stem\_cost, surfaceenergy, totalenfit

fixed\_region – Flag for determining if region 3 should be fixed and forces set to zero for a Defect structure optimization. Default value: False

forcing – Method for controlling realistic configurations. Default value: 'Concentration'. Additional value: ‘energy\_bias’, ‘chem\_pot’, ‘other’

fpbin – bin size for fingerprint function. Default value: 0.25

fpcutoff – cutoff distance for fingerprint function in Angstroms. Default value: 15.0

fusslimit – Limitation on fitness value for FUSS point to prevent optimizer from diverging. Default value: 10.0

genealogy – Flag indicating if a genealogy of the structures in a population should be written. Default value GA: True. Default value other: False

genealogytree – Flag indicating whether or not to construct a full genealogy tree during prost-processing. Default value: False.

generate\_flag – String indicating preferred method for generating initial random population. Default value: ‘box’. Available values: ‘box’, ‘sphere’, ‘random’, ‘dumbbells’, ‘random\_box’, ‘random\_sphere’.

indiv\_defect\_write – Flag indicating if atoms in region 1 and 2 for defect calculation should be written. Default value: False

isolate\_mutation – Flag indicating if mutations for a defect structure optimization should be limited to atom set 1 only. Default value: False

lammps\_keep\_files –Flag indicating if LAMMPS data, input, trajectory, and log files should be kept or not. Default value: False

lammps\_min – Command to be used to locally minimize structure in LAMMPS. Default value: None

lammps\_min\_style – Command for style to be used to locally minimize structure in LAMMPS. Default value: 'cg'

lammps\_thermo\_steps – Number of intervals between when states are written to LAMMPS trajectory output. Can be used to prevent large trajectory files. Default value: 1

large\_box\_size – Size of box to embed non-periodic cluster structure in for energy evaluation. Default value: 500 Å

lattice\_concentration – Flag indicating if the concentration of lattice atoms versus defect atoms should be calculated. Default value: False

maxgen – Maximum number of generations to be run in an optimization. Default value: 5

metropolis\_temp – Temperature for use in Metropolis acceptance criteria. Default value: 30.0

migration\_intervals – Number of generations between migrations of structures in an Island Method calculation. Default value: 5

migration\_percent – Percentage of a population to be migrated in an Island Method calculation. Default value: 0.05

mutant\_add – Flag indicating if Mutants should be added on top of population or if they should overwrite existing structures. Default value: False

mutation\_options – Types of mutations to be performed in optimization. Parameter should be given as a list of strings. Default values depend on structure type. Available options: ase\_minimization, atoms\_add, atoms\_remove, basin\_hop\_la, basin\_hop\_permute, basin\_hop\_ra\_atoms, basin\_hop\_rattle, basin\_hop\_rotate, cell\_relax\_lammps, cell\_shape, lattice\_alteration\_crystal, lattice\_alteration\_group, lattice\_alteration\_nn, lattice\_alteration\_rdrd, lattice\_alteration\_small, lattice\_alteration, move\_la, permutation\_bulk, permutation\_crystal, permutation\_crystal\_multi, permutation, quench, random\_replacement, rattle, rotation\_geo, rotation, scale\_size, swap\_int\_local, swap\_int, swap\_vacancy, swap, zp\_rotation\_fixed, zp\_rotation

mutpb – Probability of a random move occurring. Default value for GA: 0.15. Default value for other: 1.0

natoms - Number of atoms to be used in initiating simulation. Default value: sum of atoms in atomlist parameter

natural\_selection\_scheme – Scheme for selecting which individuals will survive to the next generation. Default value: best. Available options: best, cost, fuss, fuss1, fuss1r, fuss2, fussf, fussr, metropolis, multitournament, random\_pick, rank, tournament, tournament1, tournament2

nindiv – Number of individuals in a population. Default value for GA: 10. Default value for other: 1.

number\_of\_bests – Number of individuals to keep track of in best list. Default value: 100

optimizer\_type – Type of optimizer to use for optimization. Default value: ‘Random’. Available options: 'Random', 'SA', 'BH', ‘GA’

output\_format - Format for outputting fitness data to files. Default value: 'totalenergy'. Available options: totalenergy, formation\_energy, formation\_energy\_per\_int, formation\_energy2, energy\_per\_atom, fitness

pair\_coeff – Parameter for specifying pair coefficients for use in LAMMPS. No Default value.

pair\_style – Pair style to be used in LAMMPS. Default value: None. Available options: None, bop, buck, other, tersoff, eam, eam/fs, eam/cd, edip

parallel – Flag indicating whether a simulation should be conducted in parallel. Default value: False

postprocessing – Flag to determine whether to run basic post-processing analysis on optimization output. Default value: False

pot\_file – File for potential to be used in LAMMPS. Default value: None

predator – Scheme to be used to eliminate duplicate structures and run selection method. Default value: mutation\_dups. Available options: adapting, energy\_cluster, fingerprint\_niche, fitpred\_bests, fitpred\_new, fitpred, mutation\_dups, mutation\_dups\_energy, mutation\_dups\_quench, mutation\_dups\_zp

ps\_name – Pair Style name for using in LAMMPS ‘other’ style pair style. Default value: None

ps\_other – Other parameters to be specified for LAMMPS pair styles. Default value: None

quench\_max\_temp – Maximum temperature for Quench run in Kelvin. Default value: 1000

quench\_min\_temp – Minimum temperature for Quench run in Kelvin. Default value: 2

quench\_n\_steps\_1 – Number of steps to be used in first quench. Default value: 10000

quench\_n\_steps\_2 – Number of steps to be used in second quench. Default value is twice value of quench\_n\_steps\_1

quench\_step\_size – Size of step for temperature thermostatting in LAMMPS. Default value: 0.01

r\_ab – Average distance between atoms in structure. Primarily important for adding and removing atoms. Default value: 2.5

random\_loc\_start – Flag indicating if the defect should be initialized at a random location or at the center of the bulk solid file. Default value: False

random\_vac\_start – Flag indicating if a vacancy should be initialized at a random location or near the center of the bulk solid file. Default value: False

rattle\_atoms – Flag indicating if atoms should undergo additional shaking before local minimization. Useful for systems with strong potentials. Default value: False

reqrep – Required number of repetitive fitnesses in order for system to be considered converged. Default value: 10

restart – Parameter that indicates if the optimizer should load structures from a previously started simulation. Default value: False

restart\_ints – Number of atoms included in region 1 and 2 for defect simulation for loading into a restarted simulation. Default value: 0

restart\_optimizer – Flag to determine if the optimization should restart from an Optimizer file. Default value is False.

seed – random number seed generator. Default value is random number between 0 and 10

selection\_scheme – scheme for selecting and pairing individuals for crossover operations. Default value: 'tournament2'. Available options: best, cost, fuss, fuss1, fuss1r, fuss2, fussf, fussr, metropolis, multitournament, random\_pick, rank, tournament, tournament1, tournament2

sf – parameter controlling how much of material surrounding region 1 to be included in region 2. Given as a distance from the center of region 1. Default value: 1.75

size – size of volume for generating initial structure. Given as value of radius of volume. Default value: natoms^(1/3) \*r\_ab

solidcell – Cell size for the bulk solid file provided for a Defect structure optimization. Default value: Assumes distance between 1st and 3rd atom in system.

solidfile – Filename for bulk solid file for a Defect structure optimization. No default value

stem\_coeff – Alpha coefficient value for STEM cost function. Default value: None

stem\_keep\_files – Flag indicating if image files produced by STEM calculator should be kept. Default value: True

stem\_parameters – Parameters for use in calculation of STEM image. No default value

structure – Type of structure to be optimized. Available options include: Cluster, Crystal, Defect

supercell – size of supercell of bulk structure provided for defect calculation. Default value: (1,1,1)

swaplist – Flag indicating if variable atom concentrations is allowed and what they are. Default value: False or Constr\_Swaps if parameter provided.

tolerance – Fitness difference for considering a repetitive minimum fitness between generations. Default value: 0.001

tournsize – size of group used in Tournament selection scheme. Default value: 3

trackswaps – Flag indicating if the optimizer should keep track of swaps between atom types on the lattice. Default value: False

trackvacs – Flag indicating if the optimizer should keep track of vacancies on the lattice. Default value: False

vacancy\_output = Flag indicating if vacancy atoms should be written to the structure output. Default value: False

vaspcalc – Calculator for use in calling VASP with ASE. Default value: "Vasp()"

# Case Studies

Provide some examples of inputs and outputs for different cases. Maybe include the SiC, Fe, FeCr cases here?

# User Expansion

This section includes some tips and details for how to expand the code to do more. This section includes a description of some of the key class structures included in StructOpt as well as the syntax of some of the more common functions. Some basic knowledge of python programming is recommended for understanding this section.

## Individual Class Object

The individual class is the primary class for storing information on each individual in the population. The primary attributes of this class and their default values are listed below.

data

fitness=0,

index=0,

history\_index='0',

energy=0,

tenergymx=0,

tenergymin=0,

bulki=Atoms(),

bulko=Atoms(),

box=Atoms(),

pressure=0,

volume=0,

force=0,

purebulkenpa=0,

natomsbulk=0,

fingerprint=0,

swaplist=[],

vacancies=Atoms(),

swaps=Atoms()

The primary structural information for the structure being optimized is included as an ASE Atoms object in the data attribute. It can be accessed by individual[0] once the individual has been initialized.

The duplicate function in the Individual class can be used to create a copy of an individual.

The read\_individual and write\_individual functions included in the input and output functions can be used to write out the data contained in an individual class object or create a new individual from reading a file in the proper format. Please read through the read and write functions themselves for information on the specifics of the format of these files. These functions are used for restarting structures from a previous optimization.

Users can add new attributes to the individual class object if there is a specific data value that they would like to have available to all of the various other functions that will very for different individuals in a population or over a generation. Users should be sure to set default values for a new attribute in the initialization function as well as update the duplicate function, read\_individual, and write\_individual with the new attribute in order to ensure it is used properly.

## Optimizer Class Object

The optimizer class is the primary class in StructOpt for conducting the optimization. The attributes of the optimizer class are given below.

adaptbegin

adaptmultiplier

algorithm\_type

allenergyfile

alloy

ase\_min

ase\_min\_fmax

ase\_min\_maxsteps

atomlist

best\_inds\_list

bh\_steps

bh\_temp

bopcutoff

buckcutoff

buckparameters

bulkfp

calc\_method

cell\_shape\_options

constrain\_position

constrain\_swaps

convergence

convergence\_scheme

cx\_scheme

cxattempts

cxpb

CXs

debug

demin

energy\_cutoff\_factor

evalsolid

Evaluations

filename

finddefects

fingerprinting

fitness\_scheme

fixed\_region

forcing

fpbin

fpcutoff

fusslimit

genealogy

genealogytree

generate\_flag

generation

genrep

indiv\_defect\_write

isolate\_mutation

lammps\_keep\_files

lammps\_min

lammps\_min\_style

lammps\_thermo\_steps

large\_box\_size

lattice\_concentration

loggername

mark

maxgen

metropolis\_temp

migration\_intervals

migration\_percent

minfit

mutant\_add

mutation\_options

mutattempts

mutpb

Muts

natoms

natomsbulk

natural\_selection\_scheme

nindiv

number\_of\_bests

optimizer\_type

optimizerfile

output\_format

overrideconvergence

pair\_coeff

pair\_style

parallel

postprocessing

pot\_file

predator

ps\_name

ps\_other

purebulkenpa

quench\_max\_temp

quench\_min\_temp

quench\_n\_steps\_1

quench\_n\_steps\_2

quench\_step\_size

r\_ab

random\_loc\_start

random\_vac\_start

rattle\_atoms

reqrep

restart

restart\_ints

restart\_optimizer

Runtimes

seed

selection\_scheme

sf

size

solidcell

solidfile

stem\_coeff

stem\_keep\_files

stem\_parameters

structure

supercell

surfacecell

surfacefile

surftopthick

swaplist

tolerance

tournsize

trackswaps

trackvacs

vacancy\_output

vaspcalc

The parameters for the Optimizer class are initialized using the read\_parameter\_input.py file in the inp\_out subfolder of StructOpt. The primary function in the Optimizer class is the run function. This is the program that actually executes the optimization.

## Basic Algorithm Flow

## 

## Seeding a Population

Sometimes a user may want to start a given optimization with a particular set of structures beyond what is currently allowable in the standard population generation parameters. Often seeding a population with specific structures that the user believes will be fairly common or likely in the optimization can help to steer the optimization towards a specific minimum more quickly. However, seeding a population can also cause the optimization to become stuck in a certain local minimum rather than find the global minimum. To seed a population with a certain set of structures, first construct the structures and save them to separate ‘xyz’ files. The files should be labeled ‘indivXX.xyz’ where XX is a number between 00 and the number of individuals in the population. For example, if you had 5 structures you wanted to seed a population with, you would create 5 xyz files titled ‘indiv00.xyz’, ‘indiv01.xyz’, ‘indiv02.xyz’, ‘indiv03.xyz’, and ‘indiv04.xyz’. These files should then be added to a folder with the title of the filename for the optimizer output. To run an optimization with these structures, the user would then set the desired optimization parameters as usual and then set the restart flag to True. This will inform the optimizer to load the structure from the files contained in the folder rather than generate new structures. It should be noted that if the xyz file contains multiple structures, only the last structure will be read in to the population.

Additionally, if a user just wants to check the fitness of a specific structure versus some of the structures predicted by an optimization, the user can simply add this structure to the end of the ‘indiv00.xyz’ file or other individual file in the output folder.

## Adding New Fitness Functions

The current version of StructOpt includes many different types of fitness functions (see the fitness functions options section of this document for more information on the available fitness functions that are provided with StructOpt). However, users may wish to expand the code with their own types of fitness functions. StructOpt is designed on a switch system to allow it to be flexible enough to easily accommodate new user fitness functions assuming these fitness functions have a general form as described below.

import logging

def fitness\_function\_name(indiv, Optimizer):

logger = logging.getLogger(Optimizer.loggername)

#Some function for calculating a fitness value

indiv.fitness = new fitness

stro = ‘Evaluating fitness of individual = {0}’.format(indiv.history\_index)

return indiv, stro

The two inputs allowed to the user defined fitness function are an Individual class object and an Optimizer class object. The outputs must be an individual class object and a string to be written to the output file.

To use this new fitness function, the user should save this function to fitness folder in the StructOpt folder. Then to implement this function, the user should just set the fitness\_scheme parameter in the input file or dictionary to the new fitness function name. Note that the name of file that contains the new fitness function should be identical to the name of the new fitness function. This is because the fitness switch function will call this new fitness scheme by importing the following:

from structopt.fitness.new\_fitness\_function import new\_fitness\_function

Also, the name of the fitness function should not include any capital letters.

If the fitness function will include an energy evaluation, then the following section of code should be included in the fitness function.

from structopt.tools.eval\_energy import eval\_energy

from structopt.inp\_out.write\_xyz import write\_xyz

import os

cwd = os.getcwd()

try:

outs = eval\_energy(Optimizer, indiv)

except Exception, e:

logger.warn('Error in energy evaluation: {0}'.format(e), exc\_info=True)

stro = 'ERROR: Problem in Energy Evaluation'

print stro

print e

stro += '\n {0}'.format(e)

os.chdir(cwd)

f=open('problem-structures.xyz','a')

totalsol = indiv[0].copy()

totalsol.extend(indiv.bulki)

write\_xyz(f,totalsol,data='Starting structure hindex={0}'.format(indiv.history\_index))

indiv.energy = 10000

f.close()

print ' Writing structure to problemstructures.xyz file. Structure (hindex) : {0}'.format(indiv.history\_index)

print ' Setting individual energy to 10000.'

outs = [10000, starting.bulki, starting, stro]

indiv = outs[2]

stro=outs[3]

indiv.energy = outs[0]

if Optimizer.structure == 'Defect' or Optimizer.structure=='Surface':

indiv.bulki = outs[1]

if abs(indiv.energy) > Optimizer.energy\_cutoff\_factor\*(len(indiv[0])+len(indiv.bulki)):

indiv.fitness = 10000

message = 'Warning: Found oddly large energy during evaluation in structure HI={0}'.format(indiv.history\_index)

logger.warn(message)

print message

print ' Setting fitness to 10000'

if math.isnan(indiv.energy):

logger.warn('Found NAN energy structure HI={0}'.format(indiv.history\_index))

indiv.fitness = 10000

indiv.energy = 10000

This section of code will allow for the energy evaluation and correct for any errors that might appear if there is an issue with the energy evaluation. Namely, it will pass any errors that occur from the energy evaluation, it will prevent abnormally large energy structures, and it will prevent NAN energy structures from disrupting the future selection and crossover functions.

## Adding New Mutations or Moves

Users may desire to add their own types of random mutations or moves to alter structures in a specific way that may be better for a specific system. StructOpt is designed on a switch system to allow it to be flexible enough to easily accommodate new user move functions assuming these move functions have a general form as described below.

def new\_mutation(indiv, Optimizer):

#Perform some move or mutation

Optimizer.output.write(‘New Mutation performed on individual\n')

Optimizer.output.write('Index = {0}\n'.format(indiv.index))

muttype='NMut'

if indiv.energy==0:

indiv.history\_index=indiv.history\_index+'m'+muttype

else:

indiv.history\_index=repr(indiv.index)+'m'+muttype

return indiv

The inputs to any mutation must be an Individual class object and an Optimizer class object and the outputs from the mutation function must be an Individual class object. It is very important that the user specify a particular muttype and assign it to the individual’s history\_index in order to ensure proper tracking using the genealogy file. Also, users should be careful not to use a muttype that is already used by a different mutation.

In order to use this new mutation, users should save the new function in the moves folder in their version of StructOpt. Then they should add the mutation as a string to their list of mutation options in the input file or dictionary. It should be noted that the file name that contains the new mutation function must be identical to the mutation function name. It is strongly suggested that users avoid using capital letters in the name of a new mutation. Users should also keep in mind that mutations that involve energy calculations will not be able to be run using the MAST integration.

## Adding New Crossover Functions

Users may also desire to add their own types of crossovers to better work with a particular system. StructOpt is designed on a switch system to allow it to be flexible enough to easily accommodate new user crossover functions assuming these crossover functions have a general form as described below.

def new\_crossover(ind1, ind2, Optimizer):

#Perform some crossover

Optimizer.output.write(‘New crossover applied between individual {0} and individual {1}’.format(ind1.index, ind2.index))

return ind1, ind2

The inputs to any crossover function are two Individual class objects to be mated and an Optimizer class object. The outputs from the crossover must be the completed Individual class objects from the crossover. Users should be careful when writing their own crossover function, as they will likely need to include options for the forcing control parameter. For examples on how to ensure that the number of atoms and type of atoms remain constant between the individuals, users should refer to the existing crossover functions available with StructOpt.

In order to use this new mutation, users should save the new function in the crossover folder in their version of StructOpt. Then they should set the cx\_scheme to the new crossover name. In general, users should avoid using capital letters in their crossover schemes. Also, users must name the file that contains the new crossover with the same name as the crossover in order to avoid import errors with python.

## Expansions with MAST

The StructOpt-MAST interface is currently in its development stage. Some of the basic features such as running simple runs have been implemented in the interface as describe in the Submitting Through MAST section of this document. However, this section describes some of the many ways this interface may be improved.

### Submitting Mutations

In the current interface set up, energy evaluations only occur once per generation after all of the crossovers and mutations have been performed. This means that, if users are submitting their individuals to queue for energy evaluations, any mutations that involve energy evaluations cannot be performed using the MAST-StructOpt interface. This means that users will not be able to take full advantage of mutations such as quenching or basin hopping. To fix this, programmers would need to either retain some memory of the mutations performed on the structures to alter the LAMMPS/VASP input accordingly or they would need to develop a method for submitting mutations to the queue.

### Looping Queue Submissions in MAST

In the current MAST–StructOpt interface, users must run mast at least every generation. Normally this is not an issue if the user is evaluating the structures using VASP as the energy calculations will take a long time and so the actual running of mast is not overly burdening. However, if the user is only submitting relatively short LAMMPS calculations to queue each generation, then running mast continuously can be a pain. In development is a version of running mast specified by including ‘LOOPED’ in the calc\_method parameter that loops over generations internally and can reduce the burden of calling mast. The trouble with this is that the queue submissions need to be performed on the head node. The current solution is to ssh to the head node from the compute node to submit the new job. However, this may require user passwords and user names on some systems.

### Mixing LAMMPS and VASP Runs with MAST

One of the many benefits of running StructOpt with MAST is the ability to perform both LAMMPS and VASP calculations. In the current implementation of the MAST-StructOpt interface, this tradeoff between programs is accomplished through the parent-child relationships. For instance, the user can optimize a structure for a few generations with LAMMPS then pass the structures into a new child directory and optimize further with VASP. The limitation here is that the user would need to specify explicitly the number of LAMMPS and VASP runs creating a recipe file with many dependencies. This is especially annoying as often users will not know how many calculations are important for a particular system. Below are some annoyances with the current implementation of this interface.

#### Restarting with Convergence

In this implementation, the switching between LAMMPS and VASP is accomplished through the completion of small optimization. This means that the user specifies either the max\_gen or req\_rep parameter such that convergence is achieved quickly. For example, the user will ost commonly specify the system to run LAMMPS with max\_gen 50 and then run VASP with max\_gen 2. However, the user may not know how many times to loop between these two programs. Ideally, the user could specify the req\_rep parameter between both programs so that the optimization will naturally converge. One option to achieve this is to switch from the use of max\_gen parameters to n\_steps or a similar parameter. This means that switching between programs would not be done as a function of convergence but rather as a function of number of generations conducted. This would allow the convergence to be calculated separately. Therefore, the user could over estimate the number of times to perform LAMMPS and VASP calculations knowing that the program will be able to stop earlier once convergence is reached. In order to fully implement this options, the interface would need to rely on restarting from the Optimizer-restart-file rather than the simple restart from the structures as currently performed when starting children from parents. To use the optimizer-restart-file, the absolute paths to the output, population, and bests structures would need to be altered to reflect the new location in the child directory similar to the alterations performed in the structoptchecker.transfer\_restart\_files function.

#### Mixed Execution Parameter

Another option to control the looping is to do so explicitly using the calc\_method parameter. The calc\_method could be set to ‘MAST\_MIXED’ and the number of LAMMPS steps and VASP steps could be specified as additional parameters such that the checker would know to switch between the two program checkers internally. This is the preferred option as it would not require the creation of many child directories, extra output files, and generally could be conducted more smoothly. It would also allow StructOpt to handle all of the convergence internally.

### Restarting a MAST-StructOpt Optimization

Users may need to restart an optimization that was performed previously with StructOpt through MAST. Therefore, the checker would need the ability to load parameters from an input file and transfer the necessary files to the desired location.

### Submitting Fitness Functions

One of the main advantages to using MPI4PY over MAST for the parallelization was the fact that each fitness function was evaluated on a separate processor rather than just each energy calculation. This is important as sometimes users may have a more complicated fitness function that requires significant computation beyond just the energy calculation. For example, optimizations that use fingerprinting or convolution STEM calculations in their fitness evaluations could have significant computational cost if not appropriately parallelized. Therefore, it may be more prudent to have the ability to submit a fitness function to the queue rather than just the energy evaluation.

### Island Method Calculations with MAST

Another unexplored potential in MAST is the ability to replace MPI4PY in the parallelization of the algorithm using the island method scheme. This would require submitting separate parallel structural optimizations to queue, collating structures after a certain number of generations, exchanging structures between different instances, and monitoring the total convergence.

# Error Messages

This section includes a list of possible error messages, what they mean, and how to fix them. Also include information on general troubleshooting.

## General Warnings

These are the current things that I know are wrong with the code and need to be fixed:

* Lammps cell relaxation mutation must be applied to a potential with mass or pair\_coefficients in calculator parameters.
* Existence of Fit=1000 due to Mutation\_Dups or run error in population will cause greater spread in FUSS and prevent convergence
* Cannot restart a simulation that is optimizing a crystal. This is primarily due to the use of xyz files as output. These files lack cell information so the structure cannot be reloaded by reading these files.
* I don’t really know what the Alloy parameter does. So for now set this to True always.
* Lattice Alteration Group mutation can cause clusters to break apart. Need a better way to limit the scope of the move distance.
* Scale Size mutation doesn’t know what to do with a defect structure and will cause weird things to happen. Need to update for Defect option.
* Write Parameters file is slightly outdated and needs to be changed to reflect all parameters.
* For the Island Method, the optimization will continue until the next round of migrations even if this exceeds the maximum number of generations specified by max gen. This is to prevent a hanging condition as processors will not perform generations at the same speed. Therefore, Migration\_Intervals should be less than max\_gen and max\_gen should be a multiple of Migration\_Intervals to reduce extra calculations beyond desired amount.
* Multiprocessor runs on multiple nodes

## LAMMPS Execution error

WARNING : Error in energy evaluation: LAMMPS exited in <path\_to\_LAMMPS\_files> with exit code: 1.

This is a common execution error that results from an issue in the execution of LAMMPS. Most common causes of this error is an improperly setup LAMMPS input file or a highly problematic structure. When using a new potential with StructOpt, it is very important to be on the lookout for this error as typically StructOpt will ignore these structures and apply a very high fitness and energy to them (10000). It will not result in a termination of the program and the optimization can actually terminate “successfully” even if this error occurs. If an optimization run finishes very quickly or the structures don’t appear to have optimized, this error could be the cause. The best way to debug this issue is to set the keep\_LAMMPS\_files parameter to True. This will allow the LAMMPS files to be written to a folder in the output directory where the user can view them. Looking through the LAMMPS input and log files in this folder can be particularly helpful as they will include the LAMMPS error messages which may indicate issues in the way the potential file or input file was formatted for a specific potential.

Note: for a defect run if the generate\_flag includes ‘random’, the initial population of structures can be highly disordered and may cause issues in the LAMMPS execution. This issue should generally correct itself within three generations.

## Large Energy Warning

WARNING : Warning: Found oddly large energy from Lammps in structure HI= <XXX>

This error results from a very poorly structured individual that has an energy that is quite large. An energy that is “quite large” is defined as an absolute energy greater than 10 times the number of atoms. If this occurs then StructOpt will set the fitness and energy to 10,000. It is a common warning to receive early on during an optimization especially for defect runs if the generate\_flag includes ‘random’ or if the size parameter is set too small. Some mutations such as ‘random\_replacement’ can also cause this warning to appear. For the most part, StructOpt will ignore this issue and continue to try to optimize the structures. However, if the warning persists through the entire simulation, the user should start looking into the LAMMPS files to ensure the energy calculations are correct. To remove this limit, it is suggested that the user define a new fitness function with a greater limit (see the Adding New Fitness Functions section). The purpose of this limit is to prevent an error in a LAMMPS evaluation from skewing the optimization, which is particularly important in the parallel scheme.